TUGAS AKHIR - TE 141599

OPTIMASI *CROSS LAYER* UNTUK PROTOKOL *DYNAMIC SOURCE ROUTING* PADA KOMUNIKASI ANTAR KENDARAAN BERBASIS *VEHICULAR AD-HOC NETWORKS* (VANETs)

Kevianda Kamarullah
NRP 2213 100 174

Dosen Pembimbing
Dr. Ir. Endroyono, DEA.
Dr. Ir. Wirawan, DEA.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi ELEKTRO
Institut Teknologi Sepuluh Nopember
Surabaya 2017

FINAL PROJECT - TE 141599

CROSS LAYER OPTIMIZATION FOR DYNAMIC SOURCE ROUTING PROTOCOL ON VEHICULAR AD-HOC NETWORKS (VANETs)

Kevianda Kamarullah
NRP 2213 100 174

Supervisors
Dr. Ir. Endroyono, DEA.
Dr. Ir. Wirawan, DEA.

ELECTRICAL ENGINEERING DEPARTMENT
Faculty of Electrical Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2017

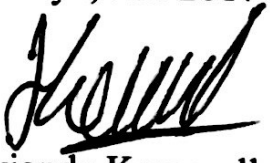
PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "**Optimasi *Cross Layer* Untuk Protokol *Dynamic source routing* Pada Komunikasi Antar Kendaraan Berbasis *Vehicular Ad-hoc Networks* (VANETs)**" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka.

Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, Juli 2017



Kevianda Kamarullah
Nrp. 2213 100 174

Halaman ini sengaja dikosongkan

OPTIMASI *CROSS LAYER* UNTUK PROTOKOL *DYNAMIC SOURCE ROUTING* PADA KOMUNIKASI ANTAR KENDARAAN BERBASIS *VEHICULAR AD-HOC NETWORKS* (VANETs)

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan Untuk
Memperoleh Gelar Sarjana Teknik Elektro
Pada
Bidang Studi Telekomunikasi Multimedia
Departemen Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui :

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Ir. Endroyono, DEA.	**Dr. Ir. Wirawan, DEA.**
NIP. 1965 04 04 1991 02 1001	**NIP. 1963 11 09 1989 03 1011**

KEMENTERIAN RISET, TEKNOLOGI, DAN PENDIDIKAN TINGGI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
FAKULTAS TEKNOLOGI ELEKTRO
DEPARTEMEN TEKNIK ELEKTRO

SURABAYA
JULI, 2017

Halaman ini sengaja dikosongkan

Optimasi *Cross Layer* Untuk Protokol *Dynamic source routing* Pada Komunikasi Antar Kendaraan Berbasis *Vehicular Ad-hoc Networks* (VANETs)

Nama : Kevianda Kamarullah
Pembimbing : Dr. Ir. Endroyono, DEA.
Dr. Ir. Wirawan, DEA.

ABSTRAK

Konsep *vehicle-to-vehicle communication* (V2V) pada *Intelligent Transportation Systems* (ITS) merupakan suatu konsep teknologi dimana antar kendaraan dapat berinteraksi satu sama lainnya. Salah satu komunikasi yang dapat menghubungkan V2V ini yaitu *Vehicular Ad-Hoc Networks* (VANETs). Pada VANET, mobil bertindak sebagai *node* dan terus bergerak sehingga membuat topologi jaringan berubah. Perubahan topologi jaringan ini mengakibatkan pengiriman data yang tidak maksimal seperti *delay* yang besar ataupun kegagalan dalam pengiriman data. Untuk itu upaya optimasi routing pada VANET ini perlu dikembangkan.

Tugas akhir ini membahas optimasi *cross layer* untuk *dynamic source routing* (DSR) pada VANET dengan mengintegrasikan *network layer dan MAC layer*. Pada *MAC layer* dilakukan proses kepadatan trafik untuk mengetahui ketersediaan kanal menggunakan *channel free time* dan mengetahui perubahan jarak antar kendaraan menggunakan *mobility prediction method*. Hasil dari kedua metode ini diteruskan ke *network layer* untuk menentukan rute terbaik. Optimasi dilakukan dengan melakukan simulasi menggunakan NS-2 yang kemudian divalidasi antara perbandingan routing tipe DSR pada VANET sebelum dan sesudah menggunakan optimasi *cross layer*.

Hasil yang didapat adalah berupa pengurangan end-to-end delay pada skenario jumlah pasangan koneksi 1, 4, dan 8 dengan pengurangan sebesar 0.01 detik, 0.015 detik, dan 0.00824 detik. Selain itu juga dapat mengurangi nilai persentase packet loss sebesar 1.3%, 0.53%, dan 0.46% pada skenario jumlah pasangan koneksi 1,4, dan 8.

Kata Kunci : *Intelligent Transportation System, Vehicular Ad-Hoc Networks, Optimasi Cross Layer*

Halaman ini sengaja dikosongkan

Cross Layer Optimization for Dynamic source routing Protokol On Vehicular Ad-Hoc Networks

Name : Kevianda Kamarullah
Advisors : Dr. Ir. Endroyono, DEA.
Dr. Ir. Wirawan, DEA.

ABSTRACT

Vehicle-to-Vehicle Communication (V2V) in Intelligent Transportation Systems (ITS) is a technology concept where nodes can interact each other. One of many types communication that can communicate V2V is called Vehicular Ad-Hoc Networks (VANETs). In VANET, cars act as node and keep moving that can change the topology of network. The changing of this network topology would make a huge amount of delay or failure of data transfer. The need of routing information in VANET is necessary.

This final project discussed cross layer optimization for dynamic source routing in VANET with the integration of network layer and MAC layer. Traffic congestion proses will be examine in MAC layer in order to give information about channel availability with a method called channel free time. Moreover, in this layer will be examine distance between vehicles with mobility prediction method. The result of these two methods directly send to network layer for giving advice to use a best route. Simulation of all the systems will use a simulator called NS-2. All of the results before the optimization and after the optimization compared each other.

The result shows that cross layer optimization can reduce end-to-end delay at the pairs of 1, 4, and 8 with the value of 0.01 sec, 0.015 sec, and 0.00824 sec. Moreover, it also can reduce the percentage of packet loss 1.3%, 0.53%, and 0.46% with the same 1, 4, and 8 pairs.

Keywords : *Intelligent Transportation System, Vehicular Ad-Hoc Networks, Cross Layer Optimization*

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Dengan mengucap puji syukur kepada Allah S.W.T., atas limpahan rahmat dan hidayah-Nya, sehingga penulis dapat menyelesaikan buku tugas akhir ini dengan judul:

Optimasi *Cross Layer* Untuk Protokol *Dynamic source routing* Pada Komunikasi Antar Kendaraan Berbasis *Vehicular Ad-hoc Networks* (VANETs)

Tugas akhir ini disusun sebagai salah satu persyaratan dalam menyelesaikan studi pada bidang studi Telekomunikasi Multimedia di Departemen Teknik Elektro, Institut Teknologi Sepuluh Nopember Surabaya.

Dalam kesempatan ini, penulis ingin menyampaikan rasa terima kasih kepada pihak-pihak yang telah mendukung penulis selama proses menyelesaikan tugas akhir ini:

1. Kedua orangtua penulis, yang selalu memberikan dukungan finansial maupun moral selama penulis menjalani proses perkuliahan di ITS, sampai akhirnya bisa menyelesaikan tugas akhir ini.
2. Kakak penulis, Adryan Fikardillah, B.Sc (Horns), M.Sc. yang telah menjadi inspirasi dan semangat penulis dalam menyelesaikan tugas akhir ini.
3. Bapak Dr. Ir. Endroyono, DEA. dan Bapak Dr. Ir. Wirawan, DEA. selaku Dosen Pembimbing atas segala bimbingan selama mengerjakan Tugas Akhir ini.
4. Bapak dan Ibu dosen departemen teknik elektro ITS, khususnya bidang studi Telekomunikasi Multimedia yang telah mendidik dan memberikan ilmunya dari awal perkuliahan sampai bisa menyelesaikan tugas akhir ini.
5. Semua rekan-rekan di lab b304 komunikasi multimedia yang telah saling bekerja dan belajar bersama selama mengerjakan Tugas Akhir ini.
6. Teman-teman Elektro ITS angkatan 2013, yang telah menjadi sahabat, keluarga, dan teman seperjuangan selama penulis berkuliah selama 4 tahun di Surabaya.

Dalam penyusunan laporan tugas akhir ini penulis menyadari masih banyak kekurangan. Oleh karena itu penulis sangat terbuka terhadap kritik dan saran untuk perbaikan karya tugas akhir ini.

Penulis berharap Buku Tugas Akhir ini dapat berguna untuk seluruh pembaca dan khususnya civitas departemen teknik elektro. Terlebih lagi, diharapkan buku ini dapat menjadi acuan dalam hal kemajuan teknologi di Indonesia.

Surabaya, Juli 2017

Penulis

DAFTAR ISI

TABLE OF CONTENT

DAFTAR GAMBAR

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB I

PENDAHULUAN

1.1 Latar Belakang

Konsep *Vehicle-to-Vehicle Communicatio*n merupakan sistem komunikasi dimana saling kendaraan dapat saling berhubungan satu sama lainnya. Tiap-tiap kendaraan ini dapat bertukar informasi seperti kecepatan, posisi, dan arah satu sama lainnya. Konsep ini memang ditujukan untuk mencegah terjadinya tabrakan antar kendaraan, dimana disinyalir dari World Health Organization (WHO), bahwa kecelakaan lalu lintas ini mengakibatkan kematian sekitar 1.2 juta manusia setiap tahunnya ditambah sekitar 50 juta manusia mengalami luka dari kecelakaan lalu lintas ini. Di Indonesia sendiri angka kecelakaan mencapai 120 jiwa perharinya.

Salah satu sistem komunikasi yang mendukung terciptanya V2V ini adalah *Vehicular Ad-Hoc Networks* (VANETs). VANET merupakan salah satu bidang pada *Mobile Ad-Hoc Networks* (MANETs), namun perbedaannya bahwa tiap kendaraan bertindak sebagai *node*. VANET memang dikembangkan untuk keamanan dan pencegahan tabrakan antar kendaraan, sedangkan MANET banyak dikembangkan pada bidang militer, pertanian, dan penanganan bencana.

Menurut Ratwani et. Al [1], *routing* pada VANET masih perlu dikembangkan, mengingat karakteristik dari VANET seperti topologi yang berubah, koneksi yang akan sering terputus, mobilitas *modeling*, *delay*, dan identifikasi lokasi. Protokol *routing* yang digunakan pada MANET tidak seluruhnya dapat digunakan untuk VANET, untuk itu teknik *routing* pada VANET perlu dikembangkan lebih dalam.

Salah satu protokol routing yang umum digunakan adalah *Dynamic source routing* (DSR), dimana protokol ini bekerja berdasarkan *routing* dari *node* sebelumnya. DSR tergolong kedalam protokol routing reactive yang hanya memilih jalur atau melakukan update jalur hanya ketika terdapat rute baru atau ketika suatu rute terputus. DSR dapat mengurangi *bandwidth overhead* dan menghindari *update routing* dengan ukuran yang besar. Akan tetapi, karena mobil-mobil yang bertindak sebagai *node* terus bergerak dan membuat topologi jaringan berubah, DSR membutuhkan waktu lebih untuk pencarian jalur sehingga nilai dari *delay* dan *packet loss* menjadi besar.

Tugas akhir ini menerapkan pendekatan *cross layer* untuk mengoptimasi *routing* tipe DSR pada VANET. Dengan melakukan pendekatan *cross layer* tersebut dapat mengurangi nilai *end-to-end delay*, dan *packet loss*.

1.2 Permasalahan

Permasalahan yang dibahas dalam tugas akhir ini adalah:

1. Bagaimana mengurangi nilai *end-to-end delay* pada VANET menggunakan metode optimasi *cross layer*?
2. Bagaimana mengurangi nilai *Packet Loss* pada VANET menggunakan metode optimasi *cross layer*?

1.3 Batasan Masalah

Pengerjaan tugas akhir ini dibatasi pada hal – hal sebagai berikut:

1. Optimasi *cross layer* mengintegrasikan *network layer* dan *MAC layer*.
2. Protokol *routing* yang ditinjau adalah *dynamic source routing* (DSR) pada VANET.

1.4 Tujuan

Adapun tujuan dari tugas akhir ini adalah sebagai berikut,

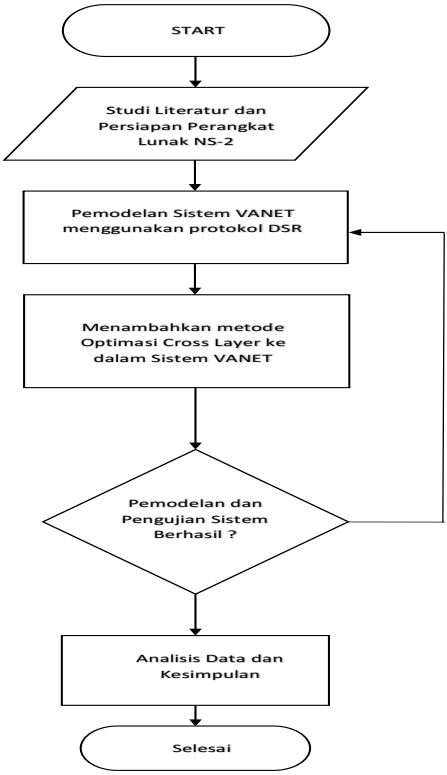
1. Penerapan Optimasi *Cross Layer* pada Protokol *Dynamic source routing* pada VANET.
2. Mengetahui cara mengoptimasi kinerja dari VANET dengan mengurangi nilai *end-to-end delay* dan *packet loss*.

1.5 Metodologi Penelitian

Penelitian ini dilakukan melalui beberapa tahapan metodologi yaitu studi literatur, simulasi VANET menggunakan *dynamic source routing protocol*, penerapan optimasi *cross layer*, dan analisis kerja sistem beserta kesimpulannya seperti yang dapat dilihat pada gambar 1.1.

1. Studi Literatur dan Persiapan Perangkat Lunak: Pada tahap ini yaitu mempelajari referensi sesuai topik dan permasalahan tugas akhir ini. Selain itu, perangkat lunak NS-2 juga dipersiapkan untuk memodelkan dan menjalankan sistem.
2. Penerapan Metode Optimasi *Cross Layer*: Menerapkan metode optimasi *cross layer* pada sistem.

3. Pemodelan Sistem: Sistem VANET dimodelkan sesuai dengan parameter yang sudah ditentukan menggunakan NS-2 dengan menjalankan protokol DSR.
4. Pengujian Sistem: Sistem dijalankan sebelum dan sesudah menggunakan optimasi *cross layer* dan kemudian dilihat apakah sesuai dengan hasil yang diinginkan.
5. Analisis Data dan Kesimpulan: Menganalisis data yang diperoleh, membandingkan hasil dari pengujian sistem sebelum dan sesudah menggunakan optimasi *cross layer*. Kemudian dapat ditarik kesimpulan dari tugas akhir yang telah dikerjakan.



Gambar 1.1 Diagram Alir Pengerjaan Tugas Akhir

1.6 Sistematika Pembahasan

Pembahasan tugas akhir ini dibagi menjadi lima bab dengan sistematika pembahasan sebagai berikut:

BAB I Pendahuluan

Bab ini meliputi latar belakang, permasalahan, tujuan penelitian, metodologi penelitian, sistematika laporan, serta relevansinya.

BAB II Tinjauan Pustaka

Bab ini berisi tentang tinjauan pustaka mengenai *Vehicular Ad-Hoc Networks* yang didalamnya terdapat penjelasan mengenai karakterisik dan protokol yang digunakan. Kemudian penjelasan mengenai *Dynamic source routing* berikut tahapan – tahapannya. Selain itu, akan dibahas juga mengenai optimasi *cross layer* secara umum dan penggabungan antara *network layer* dan MAC *layer*. Terakhir adalah pembahasan mengenai parameter pengukuran kinerja dari VANET.

BAB III Perancangan dan Simulasi Sistem

Bab ini membahas skenario penerapan optimasi *cross layer* pada VANET dan juga simulasi VANET tanpa menggunakan optimasi *cross layer*.

BAB IV Analisis Data

Bab ini berisi hasil berupa grafik dan data disertai analisis dari simulasi kinerja yang telah dilakukan.

BAB V Penutup

Bab ini berisi kesimpulan dan saran dari hasil analisis pada bab 4.

1.7 Relevansi

Hasil dari tugas akhir ini dapat diterapkan dalam mendukung konsep *vehicle-to-vehicle communication* berbasis VANET. Metode ini untuk meningkatkan performansi *routing* sehingga dapat mengurangi nilai *end-to-end delay* dan *packet loss*.

BAB 2

TINJAUAN PUSTAKA

Pada bab ini membahas mengenai tinjauan pustaka yang mendukung dalam penelitian ini. Pembahasan dimulai dengan penjelasan mengenai *vehicular ad-hoc networks* sebagai bagian dari *vehicle-to-vehicle communication*, dimana didalamnya terdapat penjelesan mengenai karakteristik dan protokol pada VANET. Selanjutnya adalah penjelasan berikut cara kerja mengenai *dynamic source routing* pada VANET. Setelah itu, akan dibahas mengenai optimasi *cross layer* secara umum dan penerapannya pada VANET itu sendiri yang dimana pada tugas akhir ini menggabungkan antara *network layer* dan *MAC layer*. Terakhir adalah pembahasan mengenai kinerja jaringan berupa *latency* dan *throughput*, efisiensi bandwidth, dan rasio pengiriman antar paket yang berpengaruh terhadap *routing overhead* dan jumlah dari *packet loss*.

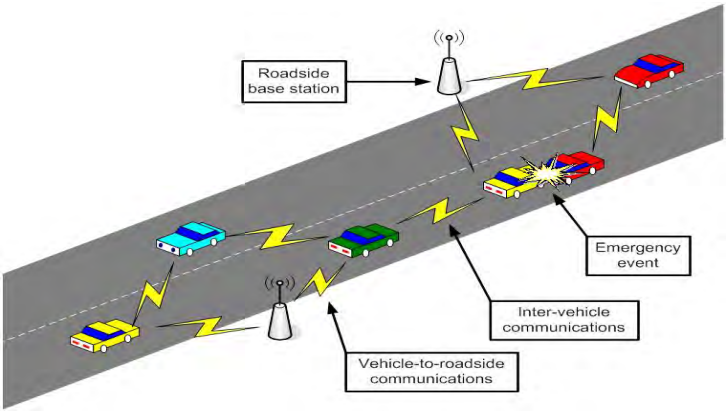
2.1 Vehicular Ad-Hoc Networks (VANETs)

Vehicle to Vehicle Communication *(V2V)* adalah konsep dimana tiap kendaraan dapat saling berkomunikasi untuk memberikan data berupa kecepatan, arah, dan posisi untuk mengurangi jumlah terjadinya kecelakaan kendaraan. Tiap kendaraan dihubungkan melalui jaringan yang disebut *Vehicular Ad-Hoc Network (VANETs)*, yang dimana komunikasi ini dapat menghubungkan banyak titik nirkabel dengan mobil bertindak sebagai *node* [2]. VANET bekerja pada spektrum frekuensi 5.9 GHz dengan bandwidth 75 MHz dan jarak 1000 m [2].

Sesuai dengan arsitektur jaringannya, VANET dapat dibagi menjadi tiga kategori. Pertama adalah *Wireless Wide Area Network* (WWAN), dimana akses poin dari gerbang seluler bersifat tetap agar dapat terhubung antara kendaraan dengan akses poin. Selanjutnya adalah *Hybrid Wireless Architecture* dimana akses poin pada WWAN digunakan pada titik tertentu. Komunikasi antara akses poin satu dengan lainnya menggunakan komunikasi ad-hoc. Kategori terakhir yaitu *Vehicle-to-Vehicle Communication* yang tidak memerlukan akses poin tetap dalam hal komunikasi antar kendaraan. Komunikasi dapat terjadi karena pada tiap kendaraan terdapat suatu unit yang dapat melakukan komunikasi dengan jaringan yang digunakan berbasis Ad-Hoc.

VANET dapat dibedakan berdasarkan tipe jalan, seperti pada jalan tol dimana mobil berjalan dengan kecepatan 80-120 km/jam dan memiliki

tingkat kepadatan yang kecil. Bentuk jalan lain yaitu pada jalanan di kota-kota besar dimana mobil berjalan dengan kecepatan 20-60 km/jam sehingga menambah tingkat kepadatan jaringan.



Gambar 1.1 Blok Diagram *Vehicle to Vehicle Communication*

2.1.1 Karakteristik Vehicular Ad-Hoc Networks

VANET memiliki karakteristik tersendiri dibandingkan dengan jaringan Ad-Hoc lainnya terutama jika dibandingkan dengan *Mobile Ad-Hoc Networks* (MANETs). Karakteristik pertama jelas bahwa VANET dirancang pada lingkungan nirkabel, untuk itu keamanan data ini harus selalu diperhatikan. Selanjutnya yaitu setiap *node* pada VANET selalu bergerak dengan mobilitas tinggi. Hal ini menimbulkan kesulitan untuk memprediksi posisi dari *node* tersebut. Berikutnya yaitu bentuk jaringan selalu berubah karena mobilitas yang tinggi dari *node – node* pada jaringan ini. Selain itu, pada jaringan ini antar *node* saling bertukar informasi sehingga menyebabkan pertukaran informasi sering sekali terjadi dengan waktu yang cepat.

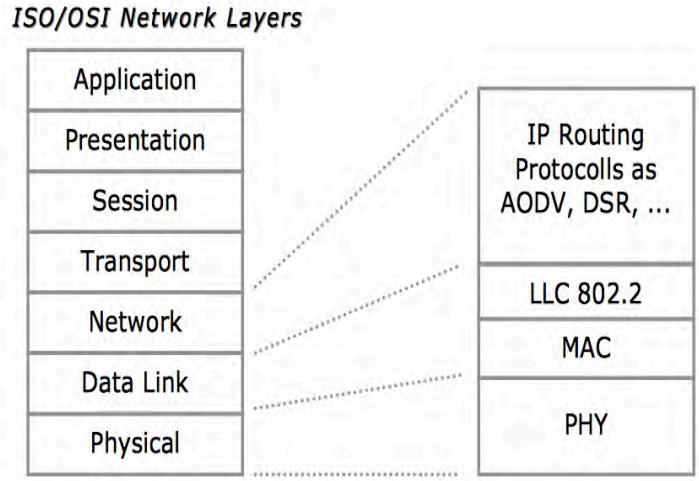
Selain dari beberapa karakteristik diatas, VANET memiliki keuntungan yang juga menjadi ciri khas dari jaringan ini. *Node – node* pada VANET tidak terlalu mementingkan sumber energi ataupun komputasi, maka dari itu daya yang digunakan saat proses transmisi adalah tidak terbatas. Ditambah lagi *node* pada VANET selalu terjaga secara fisik, sehingga mengurangi dampak dari adanya gangguan terhadap infrastrukturnya.

2.1.2 Protokol dan Penggunaan Kanal pada VANET

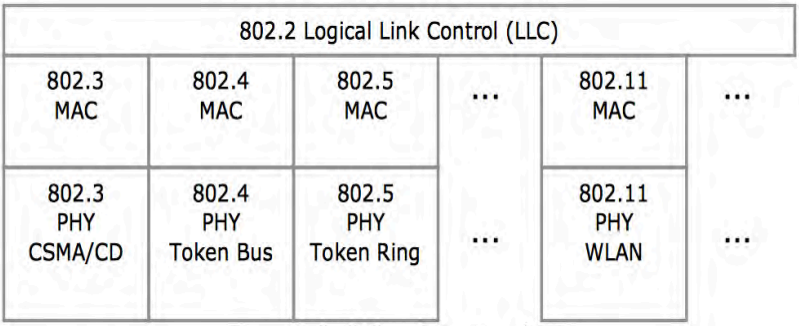
VANET menggunakan protokol MAC 802.11 dan IEEE 1609 yang ummumnya diketahui dengan *Wireless Acess in Vehicular Environments* (WAVE). Protokol ini menggunakan dua tipe kanal: *Control Channel* (CCH) dan *Service Channels* (SCH). CCH digunakan untuk penyebaran yang berkaitan dengan pengendalian informasi dan keamanan data. Sedangkan SCH digunakan untuk penyebaran informasi lainnya seperti *video streaming* [2].

IEEE 802 merupakan protokol yang menjadi dasar cara kerja dari *Local Area Networks* (LAN) dan *Metropolitan Area Networks* (MAN). Protokol 802 memiliki *logical link control* (LLC) yang di standardisasi pada 802.2. Pada bagian atas dari LLC ini yaitu *network layer* yang umumnya berisi protokol *routin*g seperti *Ad-Hoc On Demand Distance Vector* (AODV) dan *Dynamic source routing* (DSR) untuk VANET ataupun MANET. Gambar 2.2 dibawah ini merupakan urutan lapisan sesuai dengan penjelasan diatas.

Pada bagian bawah dari LLC yaitu MAC *layer* dan *Physical layer*, dimana kedua lapisan ini memiliki standar yang sama yaitu IEEE 802.11. Untuk lebih jelasnya protokol dari tiap – tiap standar dapat dilihat pada gambar 2.3.



Gambar 2.2 ISO/OSI Layer Model [3]



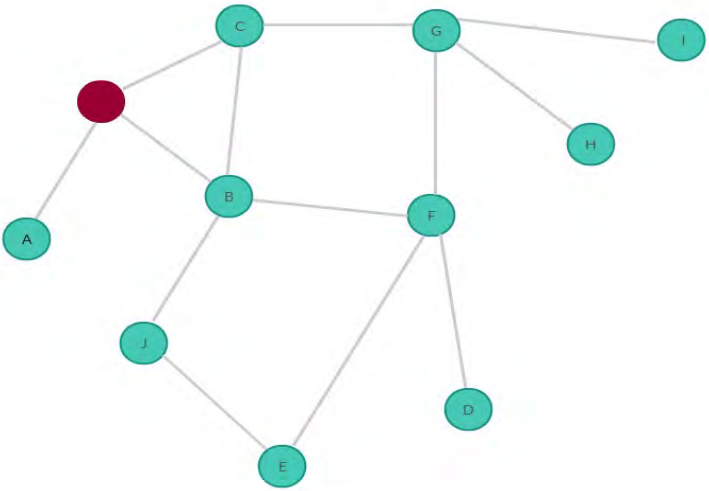
Gambar 2.3 802 LLC, MAC, & PHY [3]

Protokol IEEE 1609 digunakan untuk *Wireless Access in Vehicular Environments* (WAVE). Fokus dari protokol ini yaitu arsitektur, model komunikasi, struktur manajemen, mekanisme keamanan, dan akses fisik dengan kecepatan tinggi (27 Mb/s & 1000 m). IEEE 1609 juga membahas aplikasi yang digunakan pada WAVE, diantaranya yaitu IEEE 1609.0 untuk manajemen aktivitas dari jaringan yang didapat dari IEEE P1609.1. Untuk keamanan jaringan terdapat pada IEEE P1609.2 dan protokol untuk *network layer* pada IEEE P1609.3.

2.2 *Dynamic source routing*

Dynamic source routing Protocol (DSR) adalah salah satu protokol *routing* yang terdapat pada VANET yang tergolong kedalam *reactive routing protocol* karena bekerja berdasarkan *routing* dari *node* sebelumnya. Tiap *Node* harus memiliki tabel *routing* dan entri pada tabel *routing* akan diupdate berdasarkan perubahan topologi yang terjadi. Jaringan pada DSR ini dapat terbentuk sendiri karena tidak membutuhkan infrastruktur maupun administrasi jaringan. Setiap *node* bekerja satu sama lain untuk meneruskan paket sehingga akan tercipta komunikasi antar *node* secara otomatis dalam rentang jarak tertentu.

Pada proses pengiriman paket, *node* pengirim akan membuat *source route* pada *header* dari paket yang berisikan alamat dari setiap *node* lainnya. Hal ini bertujuan untuk memberikan alamat dari *node* penerima terdekat. Setelah itu paket akan dikirimkan dari *node* pengirim ke penerima. Jika *node* penerima tersebut bukanlah *node* tujuan, paket akan diteruskan ke *node* lainnya yang teridentifikasi pada *source route*. Ketika



Gambar 2.4 *Dynamic source routing Protocol*

paket sudah mencapai pada *node* tujuan, maka paket akan diteruskan ke layer diatasnya yaitu *network layer*.

Protokol DSR sendiri terdiri dari dua tahapan: *route discovery* dan *route maintenance*.

2.2.1 Route Discovery

Secara umum *route discovery* adalah ketika sebuah *node* akan mengirim paket ke *node* tujuan, *node* pengirim melihat tabel *routing* miliknya. Jika rute tujuan tidak ada pada tabel *routing* miliknya, maka *node* pengirim ini melakukan *broadcast packet route request*. *Packet route request* ini berisi alamat tujuan, alamat *node* sumber dan ID. *Node* lain yang menerima *packet route request* ini kemudian melihat tabel *routing* yang dimilikinya. Jika *node* perantara tidak mengetahui *node* tujuan, maka *node* tersebut akan menambahkan alamat *node* perantara tersebut kedalam paket dan kemudian meneruskannya. *Route reply* dibangkitkan ketika rute yang diminta mencapai *node* tujuan atau *node* perantara yang mempunyai informasi *node* tujuan pada tabel *routing*nya. *Node* tersebut akan menyampaikan *packet route reply* ke *node* selanjutnya hingga sampai ke *node* sumbernya.

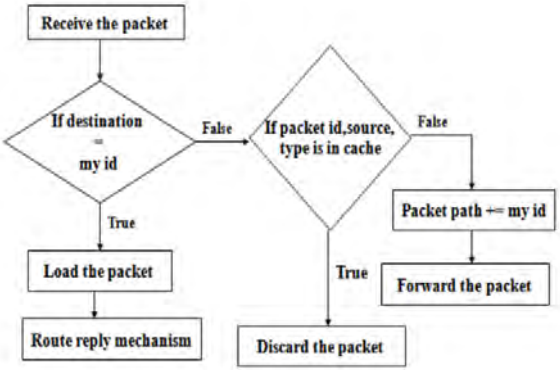
Pada jaringan ad-hoc, jika *node* sumber dan tujuan pada jangkauan transmisi yang sama, maka dalam menemukan rute hanya perlu menyebarkan permintaan rute sederhana kepada *node* tujuan. Alamat MAC akan langsung digunakan untuk mengirimkan paket dan juga tidak memerlukan *update* mengenai *routing* dalam rangka menghemat lebar pita dari jaringan [4].

Route discovery merupakan suatu teknik untuk menemukan rute seperti yang dijelaskan sebelumnya namun pada kasus ini *node* sumber dan *node* tujuan tidak berada pada jangkauan yang sama. Caranya yaitu dengan mengirimkan *request packet* (RREQ) lalu mengirimkan permintaan tersebut dengan menggunakan teknik f*looding*. Selama paket *request* tersebut dikirimkan, setiap *host* menambahkan alamat sendirinya sebelum menyebarkan paket *request*nya kepada *node* tetangga. Ketika menerima paket *request* dan *host* menemukan alamat yang sudah tersimpan di rute, maka paket *request* tersebut akan dibuang dan tidak dikirim lagi. Banyak *mobile host* berada pada jangkauan yang sama sehingga menyebabkan paket *request* yang sama. Untuk itu agar paket *request* yang sama itu dapat dihapus, setiap paket *request* memiliki *unique id* dari *node* sumber. Setiap *host* menyimpan *request id* dan alamat pengirim dari setiap paket *request* dan membuangnya jika paket tersebut sudah dikirimkan sebelumnya [4]. Hal ini bertujuan agar rute terpendek dapat dicapai. *Mobile host* harus menyimpan rute untuk digunakan dalam mengirimkan paket kepada *node* tujuan yang sama agar dapat mengurangi *routing overhead* pada protokol DSR. Jadi pada kasus ini, tidak memerlukan proses *route discovery* lagi untuk berkomunikasi antar *node*nya.

Ketika *host* sudah menerima paket *request*, maka *host* tersebut langsung memproses permintaan tersebut seperti langkah berikut:

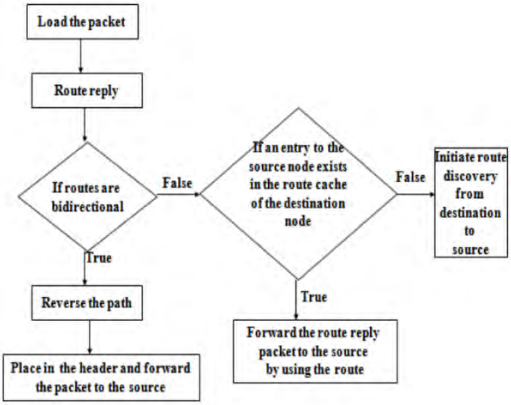
- Jika alamat *node* dan *request id* dari *node* pengirim *packet request* ditemukan karena sudah ada sebelumnya, maka paket akan dibuang dan tidak diproses lebih jauh lagi.
- Sebaliknya, jika tujuan dari *packet request* cocok dengan alamatnya, maka rute yang sudah disimpan tadi berisikan rute yang dapat digunakan untuk *packet request* sampai ke *node* tujuan.
- Sebaliknya, maka harus menambahkan alamat *host* ke rute pengiriman *packet request* dan paket dikirim ulang.

Rute tersebut digunakan untuk pengiriman paket sampai mencapai *host*.



Gambar 2.5 Diagram Alir *Route Discovery* [5]

Selanjutnya adalah mekanisme *route reply*. Untuk mengembalikan *packet reply* kepada inisiator dari *route discovery*, *host* target harus memiliki rute yang dapat digunakan untuk mengembalikan *packet reply* ini ke inisiator. Target akan mengirimkan *packet reply* menggunakan rute yang sama saat pengiriman *packet request* tadi. Sebaliknya, *node* tujuan mengambil rute dari data yang ada dan menggunakannya untuk mengirimkan *packet reply*.



Gambar 2.6 Diagram Alir Proses RREP [5]

2.2.2 Route Maintenance

Route maintenance dicapai dengan *packet route error* dan *acknowledgement* (ack). Ketika paket yang dikirim terdapat kerusakan atau bahkan datanya hilang, maka *route maintenance* akan bekerja. Pada level ini, jika *MAC Layer* memberikan laporan berupa kegagalan transmisi, maka *node* penerima akan mengirimkan *packet route error* kepada *node* pengirim pertama [5]. *Packet route error* berisikan alamat *node* yang akan mengirimkan paket dan *node* yang akan menerima paket. Ketika paket ini diterima pada *node* pengirim pertama, rute untuk mencapai *node* penerima ini akan dihapus dan rutenya diputus. Konfirmasi diterimanya packet error route ini menggunakan protokol standar yang terdapat pada MAC *layer* yaitu *link-level acknowledgement frame* yang didefinisikan berdasarkan IEEE 802.11.

2.3 Optimasi Cross Layer

Pada model OSI *layer*, terdapat batasan antar lapisan sehingga informasi disimpan secara aman pada tiap lapisannya. Akan tetapi hal ini justru berbeda dengan konsep *cross layer*, karena konsep ini justru tidak memikirkan batasan tersebut untuk membuat antar lapisan dapat saling berinteraksi satu sama lainnya. Karena itulah informasi yang terdapat pada satu lapisan dapat dikirimkan ke lapisan lainnya. Optimasi *cross layer* ini dapat memperbaiki *quality of service* (QoS) dari sebuah jaringan dengan operasi dan kondisi tertentu.

Cross layer yang akan digunakan pada tugas akhir ini adalah mengintegrasikan *network layer* dan *MAC layer*. Pada *MAC layer* akan terjadi proses estimasi kepadatan trafik data berdasarkan metode *channel free time (*CFT*)* [6]. Selanjutnya yaitu metode *mobility prediction* untuk memprediksi mobilitas dari *node* berdasarkan informasi dari *node-node* lainnya yang berdekatan dengan *node* pengirim informasi [7]. Pada *network layer*, kedua metode ini digunakan dalam pemilihan rute terbaik untuk mengirimkan *route request forwarding* berdasarkan rute dengan *delay* yang kecil dan *route error* yang kecil.

2.3.1 Medium Access Control (MAC) Layer

Medium Access Control (MAC) *Layer* merupakan lapisan kedua dalam OSI *Layer*, dimana lapisan ini berada pada lapisan *data-link*. Penambahan metode optimasi *cross layer* yang akan diterapkan pada tugas akhir ini berada pada lapisan MAC. Beberapa metode penambahan

pada lapisan MAC ini adalah *channel availability calculation*, *channel free time*, dan juga *mobility prediction* yang akan dijelaskan pada sub-bab dibawah ini.

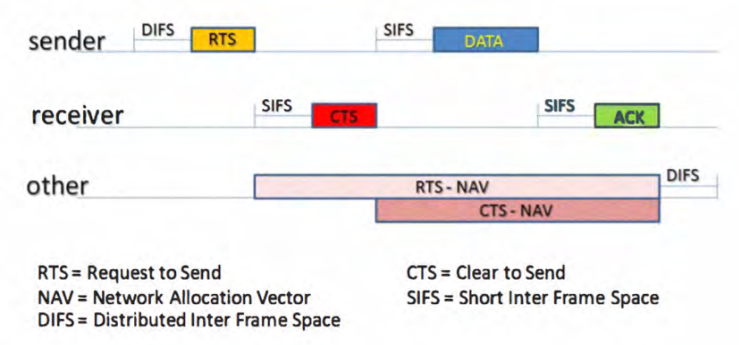
2.3.1.1 Channel Availability Calculation

Gambar 2.7 dibawah adalah diagram waktu pada IEEE 802.11 *Request-to-Send* (RTS) dan *Clear-to-Send* (CTS) pengiriman paket saat permintaan kanal dan transfer data. *Node* pengirim tidak akan tersedia setelah mengirim paket RTS sebelum paket ACK diterima dan *node* penerima tidak akan tersedia walaupun paket RTS sudah diterima namun paket ACK belum dikirim [5].

Untuk *node* lainnya, ketika paket RTS dan CTS diterima, *node-node* lainnya ini mengetahui bahwa kanal sedang digunakan untuk periode waktu tertentu. Periode waktu ini didukung berdasarkan *Network Allocation Vector* (NAV). Untuk mengetahui ketersediaan kanal dapat menggunakan rumus channel availability calculation (CAV) [7] dibawah.

$$\text{CAV} = 1 - \frac{sum\ of\ T_{busy}}{Specified\ Time\ Period} \tag{2.1}$$

Sedangkan untuk menghitung waktu sibuk (T_{busy}) yaitu menggunakan hasil kumulatif dari waktu NAV pada tiap *node* yang mengetahui waktu pengiriman pada periode waktu tertentu (*Specified Time Period*) [7].



Gambar 2.7 Diagram Waktu dari IEEE 802.11 RTS/CTS [7]

2.3.1.2 Channel Free Time

Kepadatan pada jaringan MANET menyebabkan *delay* yang lama, nilai rata – rata *packet loss* yang besar, dan juga *routing overhead*. Pergerakan dari *node* menyebabkan naiknya kepadatan jaringan. Terlebih lagi, terdapat paket *routing* yang diprioritaskan karena berisi data penting yang memang harus didahulukan sehingga menyebabkan kepadatan bertambah.

Lebar pita residual digunakan sebagai cadangan area yang padat, seperti jika area tersebut memerlukan *bandwidth* yang lebar untuk menerima data trafik dan tidak mempengaruhi komunikasi yang sedang berlangsung.

Channel Free Time (CFT) merupakan perhitungan metric yang berlangsung pada *MAC layer*. Perhitungan ini dapat dijadikan acuan dalam pengambilan keputusan untuk *routing* di *network layer* sehingga membantu dalam pemilihan rute agar terhindar dari daerah yang padat. Utilisasi dari *MAC layer* berisi 1 (padat) dan 0 (*idle*) [6]. Pada algoritma ini, terdapat parameter T_{busy} yaitu ketika media sedang dalam kondisi yang sibuk terhadap T_{mac} yaitu waktu penggunaan media nirkabel disekitar *node*. Untuk lebih jelasnya dapat dilihat pada persamaan dibawah [6]:

$$\mathrm{CFT} = 1 - \frac{\sum T_{busy}}{T_{mac}} \quad , \quad T_{busy} \subseteq T_{mac} \tag{2.2}$$

Status dari tingkat kepadatan/congestion (Cs) mengindikasikan dua keadaan, yaitu *forward* dan *drop*. *Forward* berarti paket *request* (RREQ) dapat diproses dan disebarkan kepada *node* tetangga. Sedangkan *drop* berarti tidak ada cara untuk memproses dan menyebarluaskan paket ke *node* tetangga. Lebih jelas lagi, *threshold* merupakan parameter yang diatur untuk membantu performa dari jaringan dan bernilai pada jangkauan 0 – 1.

Gambar 2.8 menjelaskan prosedur tambahan saat suatu *node* menerima paket RREQ pada protokol DSR. Ketika suatu *node* menerima paket RREQ, selain dari *node* tujuan, hal pertama yang dilakukan adalah melihat nilai Cs untuk menentukan paket RREQ tersebut dapat diteruskan atau tidak. Jika nilai Cs tersebut berada pada level *drop*, maka proses tidak dilanjutkan untuk mengurangi *routing overhead* dan memblokir masuknya data trafik. Sebaliknya, jika nilai Cs berada pada level *forward*

maka proses protokol DSR ini berjalan seperti biasanya dengan menggunakan langkah – langkah yang sudah ada.

```
When RREQ packet received.

If (It is not the destination node) and (Cs = drop level) {
  drop RREQ packet received;
}
Else {
  If (the first copy) {
    create reversed route to the source;
  }
  If (the destination node or any intermediary node has an up-to-date
route) {
    send RREP packet to the source;
  }
  Else {
    update RREQ packet and rebroadcast it to its neighbors;
  }
}
```

Gambar 2.8 Proses Tambahan Pada Penerimaan *RREQ Packet* [6]

2.3.1.3 Mobility Prediction

Pada bagian *Mobility Prediction*, diperlukan jarak antar kendaraan sebagai fungsi waktu terhadap rata-rata perubahan jarak. Untuk menemukan jarak, kuat sinyal pada paket yang diterima oleh *node* digunakan. Pengunaan *Hello Packet* sebagai paket kontrol pada protokol DSR untuk mengetahui informasi *node* yang berdekatan dan menggunakan *two-ray ground reflection* model [7] sesuai rumus

$$P = \frac{P_t G_t G_r (h_t^2 h_r^2)}{d^4} \tag{2.3}$$

$$d = \sqrt[4]{(G_t G_r h_t^2 h_r^2)\frac{P_t}{P}} \tag{2.4}$$

dimana:

P = kuat sinyal dari paket yang dikirim
P_t = kuat sinyal dari antena pemancar
G_t = *gain* antena pemancar
G_r = *gain* antena penerima
h_t = tinggi antena pemancar, h_r = tinggi antenna penerima

d = jarak antar antena pemancar dan penerima

Dari rumus diatas, akan didapat kuat sinyal dari paket yang dikirim dan waktu antar kedatangan dari paket. Selanjutnya yaitu dapat menghitung perubahan jarak [7] berdasarkan rumus:

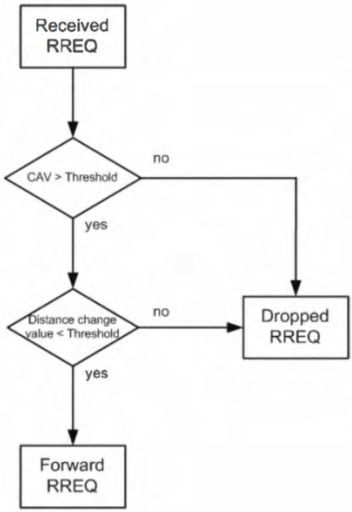
$$Distance\ change\ = \left|\frac{d_{pkt1}-d_{pkt2}}{t_{pkt1}-t_{pkt2}}\right| \quad (2.5)$$

Jika tidak ada paket yang diterima dari *node* yang berdekatan dalam rentang waktu frame tertentu, dapat diasumsikan bahwa tidak ada *node* yang berada pada daerah pengiriman sehingga dapat dihilangkan dari perhitungan jarak antar *node*. d_{pkt1} dan d_{pkt2} adalah jarak yang didapat dari paket yang diterima, sedangkan t_{pkt1}dan t_{pkt2} adalah waktu yang didapat ketika paket sudah terima. Ketika nilai dari perubahan jarak ini besar, maka *node* lain yang berdekatan sedang bergerak menjauh ataupun mendekat dalam kecepatan tinggi. Ketika nilai dari perubahan jarak ini kecil, maka *node* lain yang berdekatan sedang dalam jalur yang sama berada antara dibelakang atau didepan. Dapat dikatan bahwa, *route lifetime* akan bernilai kecil jika nilai dari perubahan jarak besar begitupun sebaliknya. Pemilihan *route lifetime* dengan nilai yang besar dapat mengurangi *delay* jaringan dan *route error*.

2.3.2 Network Layer

Pada *network layer* akan terjadi pemilihan rute untuk meneruskan paket RREQ atau tidak. Gambar 2.9 dibawah menjelaskan proses yang terjadi di dalam *network layer*. *Node* penerima akan melihat hasil dari Cs dan rata-rata perubahan jarak untuk menentukan paket akan diteruskan atau dihentikan. Hasil Cs ini didapatkan dari perhitungan proses channel free time pada lapisan dibawahnya yaitu lapisan MAC [6].

Adapun terdapat dua skenario terhadap hasil Cs yang telah dihitung pada lapisan sebelumnya. Jika hasil dari Cs memiliki nilai melebihi batas yang telah ditentukan dan rata-rata perubahan jarak memiliki nilai kurang dari batas yang telah ditentukan, maka paket akan diteruskan ke *node* yang berdekatan. Sedangkan jika hasil dari Cs justru kurang dari batas yang telah ditentukan begitupun dengan rata – rata perubahan jaraknya, maka paket tidak akan dikirimkan.



Gambar 2.9 Diagram Proses RREQ pada Network Layer [7]

2.4 Network Simulator (NS-2)

Network Simulator (NS-2) merupakan alat simulasi yang memiliki kejadian diskrit dikhususkan untuk meneliti mengenai jaringan. NS-2 menyediakan berbagai skema *routing*, protokol *multicast*, dan protokol IP seperti UDP, TCP, RTP, dan SRM melalui kabel maupun nirkabel serta satelit. Selain itu, NS-2 juga menyediakan beberapa algoritma dari *routing* dan *queuing*. NS-2 tidak hanya dapat dijalankan pada sistem operasi Linux, namun juga dapat dijalankan pada sistem operasi lain yaitu Windows dan MacOS.

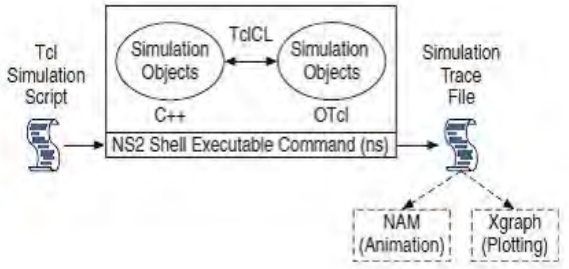
NS-2 dimulai berdasarkan varian dari simulator jaringan bernama REAL pada tahun 1989. REAL adalah alat simulasi yang ditujukan untuk mengamati perpindahan data dan perubahan bentuk jaringan pada jaringan berbasis *packet-switched*. Sampai saat ini NS-2 dikembangkan oleh VINT Group yang didukung penuh oleh *Defense Advanced Research Projects Agency* (DARPA) dengan SAMAN dan NSF melalui CONSER. Keseluruhan ini bekerja sama juga dengan peneliti lain seperti ACIRI.

Pada Network Simulator terdapat tampilan baik dengan *node* yang bergerak maupun *node* yang tidak bergerak. Paket – paket yang membangun dalam simulasi jaringan ini antara lain:

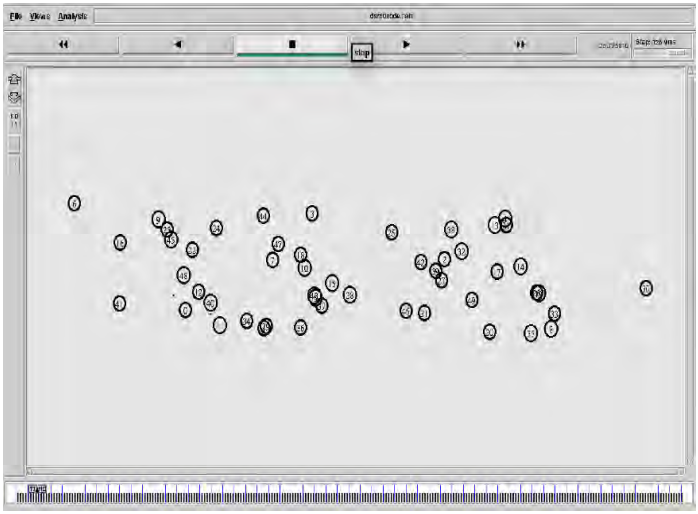
- Tcl : *Tool Command Language*
- Tk : *Tool Kit*
- Otcl : *Object Tool Command Language*
- Tclcl : *Tool Command Language / C++ Interface*
- NS2 : *Network Simulator* versi 2
- NAM : *Network Animator*

NS-2 menyediakan suatu perintah ns, yang merupakan perintah yang dapat dieksekusi (*executable*) oleh penggunanya. Dalam menjalankan simulasi pada NS-2, perintah ns tersebut membutuhkan argumen masukan berupa nama dari skrip simulasi tcl yang telah dibuat sebelumnya. NS-2 akan menjalankan simulasi berdasarkan skenario yang terdapat pada skrip file simulasi tcl tersebut. Simulasi ini akan menghasilkan suatu *file trace* yang berisikan data hasil simulasi. File tersebut akan digunakan sebagai dasar dalam menampilkan grafik hasil simulasi dan menampilkan animasi simulasi.

NS-2 sendiri menggunakan Bahasa TCL dan C++ untuk membuat skrip programnya. TCL digunakan untuk membuat program berdasarkan objeknya, dimana bahasa ini umumnya untuk membuat struktur dan topologi jaringan. Selain itu, TCL juga lebih mudah digunakan daripada C++ dalam hal konfigurasi parameter yang dibutuhkan pada suatu jaringan. Akan tetapi, bahasa TCL ini kurang memiliki skema dan arsitektur protokol sehingga kurang cocok sebagai bahasa tunggal dalam hal penelitian. Untuk C++ sendiri adalah bahasa pemrograman yang paling banyak digunakan pada NS-2 dan merupakan bagian kernel dari NS-2 itu sendiri. Bahasa C++ ini dapat menjalankan dan melihat jalannya paket dari pengirim ke penerima ataupun dapat melihat kerja dari tiap – tiap *node*.



Gambar 2.10 Arsitektur Dasar NS-2



Gambar 2.11 Tampilan NAM

Ketika simulasi berakhir, NS membuat satu atau lebih keluaran *file* berbasis *text* yang berisi *detail* simulasi jika dideklarasikan pada saat membangun simulasi. Terdapat dua jenis file keluaran NS, yaitu : *File Namtrace*, yaitu keluaran berupa grafis simulasi yang disebut *network animator* (NAM) dan *file trace* yang digunakan untuk analisa numerik.

Akan tetapi, dari pengembang network simulator sendiri sekarang sudah memiliki *Network Simulator* versi 3 (NS-3). Namun, NS-3 mendapatkan kritik karena protokol yang didukungnya tidak sebanyak pada NS-2. NS-3 tidak mendukung protokol untuk penelitian mengenai jaringan sensor nirkabel, MANET/VANET, dan lainnya. Ditambah lagi, NS-3 tidak dapat menjalankan program yang telah dibuat pada NS-2 sehingga banyak peneliti mengenai jaringan sensor nirkabel dan VANET masih menggunakan NS-2.

2.4.1 Protokol dan Model pada NS-2

Adapun beberapa model yang dapat dibuat dan dijalankan pada NS-2 adalah sebagi berikut:

- Jaringan Kabel dan Nirkabel
- *Mobile Network Ad-Hoc Routing* dan *Vehicular Ad-Hoc Network Routing*

- Proses Antrian seperti *Drop-Tail*, RED
- *Quality of Service* (QoS)
- Jaringan Sensor Nirkabel
- Pemodelan Trafik

2.4.2 Transport Agent pada NS-2

Pada jaringan internet terdapat empat lapisan komunikasi TCP/IP, yaitu: lapisan aplikasi, lapisan transport, lapisan IP dan jaringan. Lapisan Transport merupakan lapisan komunikasi yang mengatur komunikasi data yang digunakan oleh lapisan aplikasi diatasnya. NS mensimulasikan lapisan transport dengan objek simulasi bernama *transport agent*. Pada simulasi pengiriman data, *transport agent* tidak dapat berdiri sendiri sehingga membutuhkan lapisan aplikasi di atasnya yang berfungsi sebagai pembangkit trafik.

Adapun protokol lapisan *transport data* yang didukung NS-2 sebagai berikut:

1. **TCP (Transport Control Protocol)**

   Transport Control Protocol merupakan protokol *transport* yang andal karena mempunyai mekanisme yang dapat memastikan paket dapat diterima oleh *client*. Pada saat TCP sedang dikirimkan ke *node* penerima, protokol ini akan memberikan *state acknowledgement* (ACK). Jika ACK ini tidak diterima oleh *node* pengirim, maka secara otomatis ACK ini akan dikirim ulang dalam selang waktu tertentu.

2. **UDP (User Datagram Protocol)**

   User Datagram Protocol adalah lapisan *transport* yang tidak andal, *connectionless*, dan merupakan kebalikan dari lapisan *transport* TCP. Setiap aplikasi *socket* dapat mengirimkan paket – paket berupa datagram. Istilah datagram diperuntukkan terhadap paket dengan koneksi yang tidak andal. Koneksi tidak andal adalah koneksi yang tidak akan mengirimkan keterangan saat pengiriman data gagal.

   UDP tidak menjamin kevalidan data saat data sampai ke *node* penerima. Datagram yang sampai mempunyai kemungkinan tidak sampai, rusak, duplikasi atau hilang tanpa diketahui penyebabnya. Untuk itu penggunaan UDP lebih tepat saat data – data yang dikirim merupakan data kecil dalam jumlah banyak. Dengan itu, maka data akan sampai lebih cepat dan lebih efisien mengingat sifat UDP yang tidak akan melakukan pengecekan terhadap paket tersebut.

3. RTP (Real Time Transport Protocol)

RTP menyelenggarakan *end to end delivery services* untuk data yang memiliki karakteristik *real time*, seperti VoIP dan video interaktif. Layanan tersebut termasuk identifikasi tipe *payload*, pengurutan, *timestamping*, dan monitor pengiriman data.

2.4.3 Level Aplikasi pada NS-2

Terdapat dua tipe dasar aplikasi pada NS-2, diantaranya:

1. Simulated Application

Simulated Application merupakan aplikasi yang dapat langsung disimulasikan pada NS-2, yaitu FTP dan Telnet. FTP dibangun untuk mensimulasikan *bulk data transfer*. Sedangkan Telnet adalah aplikasi yang diatur oleh transport agent dengan jumlah paket yang ditransmisikan diatur oleh mekanisme *flow control* dan *congestion control* TCP.

2. Generator Traffic

Pembangkitan trafik pada NS-2 terdiri atas empat tipe yang diantaranya Eksponensial, *Pareto*, CBR, dan *Traffic Trace*. Eksponensial adalah trafik yang dibangkitkan pada waktu kedatangan antar paket sesuai dengan fungsi eksponensial. Sedangkan *pareto* adalah trafik yang dibangkitkan pada waktu kedatangan antar paket sesuai dengan fungsi *pareto*. Selain kedua fungsi tadi, terdapat juga trafik CBR yaitu trafik yang dibangkitkan secara kontinyu dengan *bit rate* konstan dan *traffic trace* yang merupakan trafik dari sebuah *file trace*.

2.5 Parameter Kinerja dari Vehicular Ad-Hoc Networks

Quality of Service (QoS) adalah kemampuan suatu jaringan untuk mencapai kinerja maksimum yang ditentukan oleh parameter – parameter tertentu. Secara khusus pada sebuah jaringan, bahwa kemampuan untuk mengatasi permasalahan seperti kepadatan trafik, waktu pengiriman yang besar, dan permasalahan lainnya agar suatu informasi dapat dikirimkan dari pengirim ke penerima.

Sesuai dengan standar IEEE 802.11 bahwa parameter untuk mengukur kinerja jaringan terdapat pada *MAC Layer*. Parameter tersebut tidak jauh berbeda dengan parameter pengukuran pada jaringan ad-hoc

secara umum. Parameter yang diperlukan antara lain yaitu *latency*, efisiensi lebar pita, dan rasio paket yang dikirim [8].

2.5.1 Latency

Latency adalah durasi waktu antara mengirim informasi dari pengirim sampai ke diterima oleh penerima. *Latency* dipengaruhi oleh kecepatan dari media transmisi (kabel koaksial, fiber optik, gelombang radio) dan *delay* transmisi oleh komponen jaringan seperti router, modem, hub, dan lainnya. *Latency* dan *throughput* adalah dasar dari pengukuran kinerja suatu jaringan. *Latency* mengukur waktu yang dibutuhkan dari mulai pengiriman sampai selesai, sedangkan *throughput* adalah waktu keseluruhan dari seluruh proses yang dalam waktu tertentu [8].

$$Throughput = \frac{\text{jumlah paket yang dikirim}}{\text{waktu yang digunakan untuk pengiriman paket}} \quad (2.6)$$

Pengiriman data dengan paket berukuran besar mengakibatkan nilai *throughput* yang besar dibandingkan pengiriman data dengan paket berukuran kecil karena *latency* bernilai besar. Jika data dikirim secara kontinyu, nilai *latency* memiliki dampak yang kecil terhadap nilai *throughput*. Akan tetapi jika pengirim menunggu *acknowledgment* (ack) sebelum paket berikutnya dikirim, maka tingginya nilai *latency* dapat menurunkan nilai *throughput*.

2.5.2 Efisiensi Lebar pita

Estimasi dari penggunaan lebar pita memiliki pengaruh yang cukup signifikan terhadap performansi dari sistem. Jika estimasi dari lebar pita lebih rendah daripada kapasitas dari suatu jaringan, maka ketersediaan lebar pita menjadi dibawah perkiraan. Begitupun jika estimasi lebar pita lebih tinggi daripada kapasitas suatu jaringan, maka ketersediaan lebar pita menjadi melebihi atau berlebihan. Dampaknya yaitu performansi sistem menjadi tidak maksimal karena estimasi lebar pita yang tidak sesuai.

Pada VANET, penggunaan lebar pita lebih banyak daripada jaringan nirkabel lainnya mengingat *node* selalu bergerak dengan mobilitas tinggi. Untuk itu salah satu faktor yang perlu diperhatikan dalam merancang VANET adalah kemampuan kendaraan atau *node* untuk bekerja sesuai dengan bentuk jaringannya [8].

2.5.3 Rasio Pengiriman Paket

Rasio pengiriman paket (PDR) merupakan perbandingan jumlah paket yang diterima oleh *node* tujuan dengan jumlah paket yang dikirim oleh *node* pengirim. Nilai dari rasio ini dipengaruhi oleh beberapa faktor seperti ukuran paket, jarak pengiriman, dan mobilitas dari *node*. Jika paket terkirim sempurna mengindikasikan bahwa penerima menerima seluruh paket sebelum waktu periode habis. Sebaliknya jika paket tidak terkirim sempurna mengindikasikan bahwa terdapat paket yang hilang.

$$\text{PDR} = \frac{\text{paket yang diterima}}{\text{paket yang dikirim}} \tag{2.7}$$

Hal mendasar dari perhitungan rasio pengiriman paket yaitu memilih rute yang benar. Rute tersebut membutuhkan *lifetime* yang lebih lama dan sedikit hop. Akan lebih baik jika *node* pengirim memiliki informasi tentang rute yang akan dipilih daripada menggunakan rute terpendek. Hal ini karena rute terpendek tersebut pasti banyak digunakan sehingga menyebabkan routing overhead [8].

Halaman ini sengaja dikosongkan

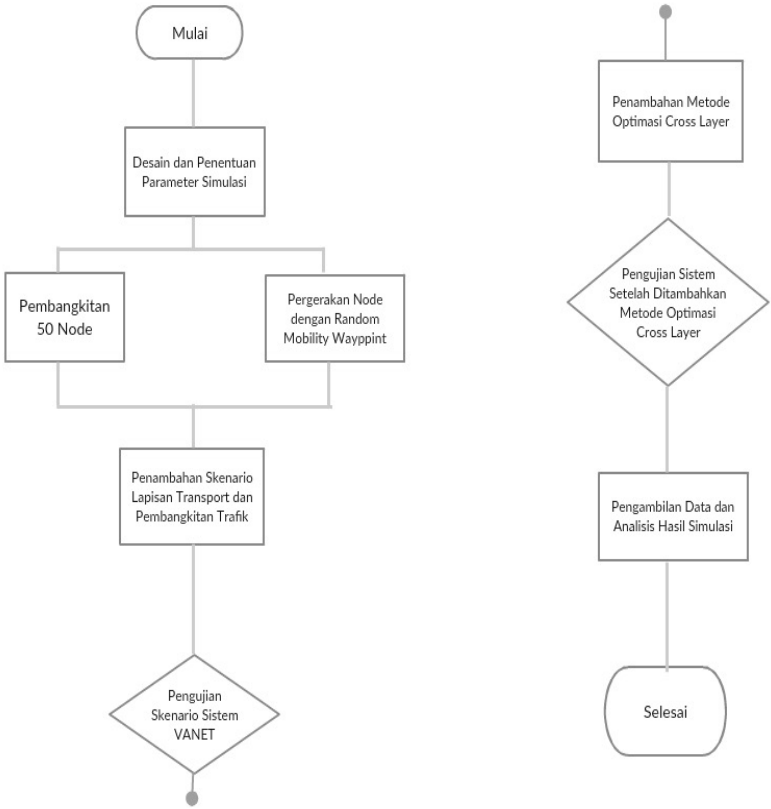
BAB 3

PERANCANGAN DAN SIMULASI SISTEM

3.1 Pendahuluan

Pada bab 3 ini akan dijelaskan mengenai perancangan dan simulasi sistem mengenai *dynamic source routing* pada komunikasi antar kendaraan dengan menggunakan perangkat lunak *network simulator* 2 (NS-2).

=



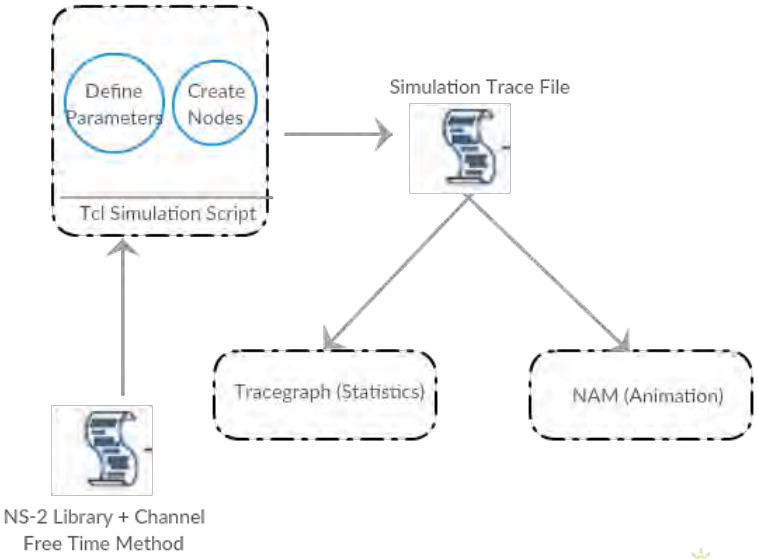
Gambar 3.1 Diagram Alir Perancangan dan Simulasi Sistem

Secara umum, perancangan dimulai dengan mempertimbangkan skema *node* bergerak berikut parameter – parameternya yang harus ditentukan terlebih dahulu. Acuan dari perancangan sistem ini berdasarkan dari jurnal referensi yang ada. Setelah itu, penulis menjalankan simulasi DSR pada VANET sebelum dilakukan optimasi *cross layer* dan setelah diberikan optimasi *cross layer*. Setelah berhasil, analisis dilakukan dengan menggunakan bantuan perangkat lunak tracegraph dan akan didapatkan kesimpulan.

3.2 Desain Simulasi *Dynamic source routing* pada NS2

Sebelum menjalankan simulasi pada tugas akhir ini, terlebih dahulu yang harus dilakukan adalah mendesain simulasi yang akan dijalankan. Dibawah ini merupakan skema dari simulasi yang akan dijalankan pada tugas akhir ini.

Dapat dilihat model sistem secara umum untuk tugas akhir ini dimana langkah mendefinisikan parameter dan pembuatan *node* sesuai dengan tabel 3.1. Kedua langkah ini ditulis dengan menggunakan C++ yang disimpan dalam bentuk *file* Tcl agar dapat dijalankan oleh NS-2.



Gambar 3.2 Model Sistem Simulasi Menggunakan NS-2

Hasil dari simulasi ini terbagi menjadi dua yaitu hasil berupa data dan hasil animasi untuk melihat kejadian – kejadian yang terjadi dari sistem yang sudah dirancang. Hasil berupa data dilihat dengan menggunakan aplikasi tracegraph sedangkan hasil animasi dilihat dengan menggunakan aplikasi NAM.

Penambahan optimasi *cross layer* dalam hal ini yaitu metode *channel free time* dilakukan pada *file* mac802-11.cc dan mac802-1.h pada folder mac di perpustakaan NS-2. Setelah metode ini berhasil ditambahkan, simulasi kembali dilakukan dengan parameter dan waktu simulasi yang sama.

3.2.1 Penentuan Parameter Simulasi

Pada bagian ini akan menjelaskan mengenai pemilihan parameter berikut dengan pembangkitan *node* pada simulator NS-2. Pada *file* Tcl yang dibuat, penulisan parameter ini ditulis pertama kali baru kemudian mendefinisikan *node – node* yang akan digunakan untuk simulasi. Pada tugas akhir ini parameter simulasi dapat dilihat pada tabel 3.1 dibawah.

Tabel 3.1 Parameter Simulasi VANET pada NS-2

Parameter	***Type***
Channel	*Wireless Channel*
Radio Propagation Model	*TwoRay Ground*
Network Interface	*Network/WirelessPhy*
MAC	*IEEE 802.11*
Interface Queue	*CMUPriQueue*
Antenna	*Omni Antenna*
Routing Protocol	DSR
Simulation Time	500 s
Node Type	*Mobile*
Node Speed	0 m/s – 36 m/s (0 km/jam – 130 km/jam)
Number of Node	50
X Dimension of Topography	1500
Y Dimension of Topography	300

Penjelasan mengenai penentuan parameter diatas adalah sebagai berikut:

1. Pemilihan kanal pada simulator NS-2 yaitu kanal nirkabel karena jaringan ad-hoc berbasis nirkabel dengan model propagasi yaitu *TwoRayGround*. Selain itu, simulasi ini juga menggunakan model antena omni.
2. Simulasi dilakukan dengan tipe jaringan antarmuka pada lapisan fisik dengan menggunakan IEEE *MAC Layer* 802.11. Pemilihan IEEE 802.11 ini disesuaikan dengan standar IEEE untuk jaringan *Mobile Ad-Hoc Network* (MANET) dan *Vehicular Ad-Hoc Network* (VANET).
3. Model antrian antarmuka menggunakan CMUPriQueue yang memang khusus digunakan untuk protokol *routing* DSR.
4. Protokol *routing* yang disimulasikan adalah *Dynamic source routing* dengan menggunakan 50 *node* dengan luas daerah topografi simulasi seluas 1500 x 300.
5. Koneksi antara *node* sumber dan tujuan berupa TCP serta diberikan trafik berupa Constant Bit Rate (CBR) pada simulasi ini.
6. Kecepatan dari pergerakan *node* ditentukan bervariasi dalam jangkauan kecepatan mobil pada jalan raya, yaitu 0 m/s – 36 m/s (0 km/jam – 130 km/jam).

3.2.2 Perhitungan Parameter Propagasi dengan Model Propagasi TwoRay Ground

Selain dari parameter simulasi diatas, terdapat parameter lain yang harus ditentukan nilainya terlebih dahulu. Nilai tiap parameter tersebut dapat dilihat seperti tabel 3.2 dibawah. Untuk menentukan nilai dari tiap parameter ini, penulis menggunakan file threshold.cc pada NS-2 dengan cara memasukkan command seperti berikut

```
kevianda@ubuntu: ~/ns-allinone-2.35/ns-
2.35/indep-utils/propagation$ ./threshold –m
TworayGround 300
```

Pada file threshold.cc ini terdapat rumus perhitungan model TwoRay Ground untuk mendapatkan nilai parameter – parameter pada tabel 3.2. Model Propagasi yang diimplementasikan pada ns digunakan

untuk memprediksi kuat daya yang diterima. Pada layer physical dari setiap node terdapat threshold penerimaan paket. Ketika paket diterima dan kuat daya tersebut bernilai lebih rendah aripada threshold, maka pengiriman paket tersebut akan terjadi error dan didrop oleh MAC layer. Pada tugas akhir ini digunakan model propagasi two-ray ground.

Model propagasi two-ray ground mempertimbangkan jalur line of sight (LOS) dan juga jalur pantulan dari tanah. Dari parameter kedua jalur tersebut terdapat panjang gelombang (lambda), namun nilai dari lambda tersebut saling menghilangkan sehingga pada model propagasi two-ray ground ini nilai dari panjang gelombang tersebut tidak digunakan.

$$P_r = \frac{P_t\, G_t\, G_r\, (h_t^2\, h_r^2)}{d^4\, L} \tag{3.1}$$

Dimana:
P_r = Daya terima antena penerima
P_t = Daya pancar antena pemancar
G_t = Gain antena pemancar
G_r = Gain antena penerima
h_t = tinggi antenna pemancar
h_r = tinggi antenna penerima
d = jarak antar kedua antena
L = faktor rugi – rugi sistem, dengan $L \geq 1$

Tabel 3.2 Parameter Propagasi pada NS-2

Parameter	***Type***
Transmit Antenna Gain	1
Receive Antenna Gain	1
System Loss Factor	1.0
Frequency	9.14 GHz
Bandwidth Data Rate	11 Mb
Transmit Power	0.281838
Receive Power Threshold	1.764149e-10
Data Frames Rate	1 Mb
Control Frames Rate	1 Mb

3.2.3 Pembangkitan dan Pergerakan Node

Jumlah *node* yang akan digunakan dalam simulasi ini ditentukan sebelumnya pada tahap penentuan parameter. Baru setelah jumlah *node* diketahui, maka letak dan juga pergerakan *node* harus didefinisikan. Letak dari *node* maupun pergerakannya didefinisikan menggunakan koordinat kartesius pada sumbu x, sumbu y, dan sumbu z.

Pada tugas akhir ini menggunakan jumlah *node* sebanyak 50 *node*. Langkah pembangkitan 50 *node* dan pergerakannya dapat didefinisikan melalui bantuan program yang sudah ada pada perpustakaan NS-2 file setdest.cc. File ini dapat mendefinisikan jumlah *node* sesuai dengan input yang kita inginkan dan juga dapat mendefinisikan pergerakan dari tiap *node* tersebut secara acak. Pergerakan acak dari *node* tersebut dinamakan dengan random waypoint mobility.

Untuk menambahkan mobilitas pada simulasi ini langkah pertama adalah dengan mengarahkan *prompt* ke subfolder setdest pada folder ns-allinone-2.35. langkah selanjutnya adalah memasukkan command prompt berikut.

```
kevianda@ubuntu: ~/ns-allinone-2.35/ns-
2.35/indep-utils/cmu-scen-gen/setdest$
./setdest -n 50 -p 0 -M 36 -t 800 -x 1500 -y
300 > dsr50nodemob.tcl
```

dimana:
n = jumlah *node*
p = waktu pause
M = kecepatan maksimal (m/s)
t = waktu simulasi
x = luas topografi pada koordinat x
y = luas topografi pada koordinat y

```
Source dsr50nodemob.tcl
```

3.2.4 Koneksi Lapisan Transport dan Pembangkitan Trafik

Dalam pembuatan koneksi di lapisan *transport* ini adalah dengan menuliskan list program pada file Tcl. Koneksi ini dibuat dengan

menghubungkan *node* yang akan bertindak sebagai sumber dengan *node* yang bertindak sebagai *node* tujuan secara manual. *Node* sumber dituliskan sebagai *node* yang bertindak sebagai TCP/UDP dan *node* tujuan dituliskan sebagai *node* yang bertindak sebagai *Sink*.

```
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
$ns attach-agent $node_(1) $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $node_(2) $sink
$ns connect $tcp $sink
```

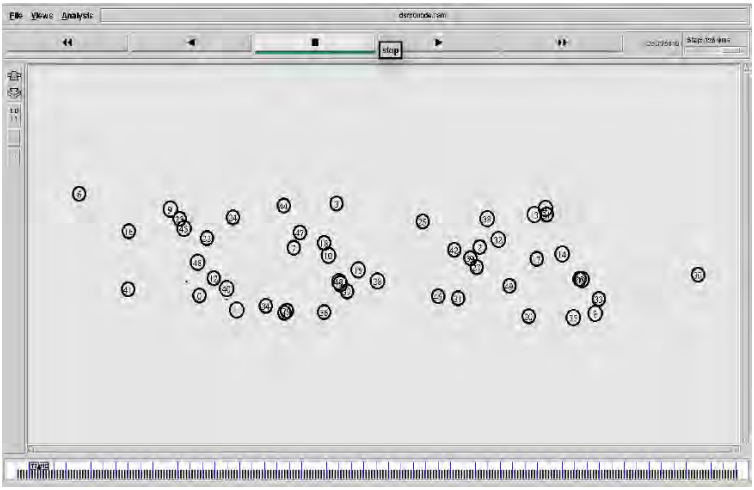
Karena lapisan *transport* tidak dapat berdiri sendiri, maka perlu didefinisikan trafik yang dihubungkan dengan TCP/UDP. Trafik dibangkitkan dengan menuliskan *list* program pada *file* Tcl berikut.

```
# Create CBR Traffic source and attach it to TCP
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 500
$cbr set type_ CBR
$cbr set interval_ 0.5
$cbr attach-agent $tcp
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
```

3.3 Skenario Simulasi

Simulasi ini dilakukan dengan menggunakan delapan skenario simulasi. Parameter yang sudah ditentukan pada sub-bab sebelum ini digunakan pada seluruh skenario simulasi ini. Namun, nilai yang diubah – ubah adalah jumlah koneksi *node* sumber dan *node* tujuan. Pada tugas akhir ini, jumlah koneksi divariasikan dari 1, 2, 4, 6, 8, 10, 12, dan 14 dengan bentuk jaringan ini saat akan dijalankan simulasi seperti gambar dibawah.

Skenario dengan 1 pasang koneksi dipilih untuk memudahkan dalam melihat cara kerja dari protokol routing DSR. Kemudian variasi pasangan koneksi divariasikan dari 2 sampai dengan 14 pasang koneksi dengan mengacu kepada referensi [6] karena jumlah tersebut dapat merepresentasikan parameter pada VANET dengan mobilitas tinggi.



Gambar 3.3 Tampilan Simulasi pada layar NAM

3.3.1 DSR 50 *Node* dengan 1 Hubungan Koneksi

Simulasi ini dijalankan hanya menggunakan 1 hubungan koneksi saja antara *node* sumber dan *node* tujuan. Hal ini diperlukan untuk membuktikan bahwa penginputan koneksi dan juga trafiknya dapat dilakukan secara manual sesuai dengan langkah pada subbab 3.2.4 yang sudah dijalankan sebelumnya.

Dengan menginputkan koneksi secara manual, maka pemilihan *node* sumber dan *node* tujuan juga dapat ditentukan oleh penulis. Pada skenario ini dipilih *node* 0 adalah *node* sumber sedangkan *node* 1 bertindak sebagai *node* tujuan. *Node – node* lainnya bertindak sebagai *node* relay yang tugasnya adalah meneruskan paket dari *node* sumber kepada *node* tujuan jika *node* relay tersebut berada pada lintasan yang digunakan dalam pengiriman paket.

3.3.2 DSR 50 *Node* dengan Lebih dari Satu Koneksi

Untuk memvariasikan jumlah koneksi ini menggunakan program cbrgen.tcl. Hanya saja inputan dari –mc [maximum connection] divariasikan sesuai dengan skenario yang telah ditentukan. File cbrgen.tcl ini terdapat pada perpustakaan NS-2 pada folder cmu-scen-gen. File ini memang secara khusus digunakan untuk membuat hubungan koneksi antara *node* sumber dan *node* tujuan.

Dibawah ini adalah langkah penulisan command prompt yang dituliskan pada layar terminal linux Ubuntu.

```
kevianda@ubuntu: ~/ns-allinone-2.35/ns-
2.35/indep-utils/$cmu-scen-gen/ns cbrgen.tcl -
type cbr –nn 50 –seed 10.0 –mc [max. connection
(2, 4, 6, 8, 10, 12, 14] –rate 1 > cbr50node.tcl
```

dimana:
type = tipe trafik yang dipilih (cbr/ftp/poisson)
nn = jumlah *node*
mc = nilai maksimal koneksi yang dibuat
rate = nilai data rate

Setelah itu, file yang telah dibangun ini harus dihubungkan dengan file tcl dsr50*node*.tcl untuk menjalankan program yang sudah ada pada NS-2. Seperti halnya dalam menghubungkan mobilitas *node* dengan file tcl nya, maka pada file tcl yang sudah ada hanya perlu menambahkan program seperti berikut. Untuk setiap koneksi yang digunakan menggunakan file yang berbeda. Dibawah ini adalah format umum untuk menghubungkan file trafik ke file tcl.

```
Source [namafiletrafik.tcl]
```

Perlu diketahui bahwa setiap *node* sumber mengirimkan informasi dan trafik yang dibangkitkan. Adapun, trafik yang digunakan adalah CBR dengan parameter dari trafik tersebut adalah

Tabel 3.3 Parameter Trafik CBR

PacketSize	512 bytes
Interval	1 sec
random	1
Max. packets	10.000 packets

3.3.3 Optimasi Cross Layer pada DSR

Pada folder instalasi NS-2, terdapat beberapa sub folder yang berisi utilitas penggunaan simulator NS-2 ini. Pada folder mac terdapat beberapa protokol lapisan mac dimana salah satunya adalah IEEE 802.11 yang digunakan pada tugas akhir ini.

Metode *Cross layer* yang pertama adalah menambahkan perhitungan waktu kosong pada kanal/*channel free time* (CFT) ke dalam *list* program dari IEEE 802-11.cc, 802-11.h pada sub folder mac. Penambahan perhitungan CFT ini akan memberikan parameter Cs yaitu status dari tingkat kepadatan kanal [6]. Perhitungan CFT tersebut dapat dilihat pada rumus dibawah.

$$\text{CFT} = 1 - \frac{\sum T_{busy}}{T_{mac}} \quad , \quad T_{busy} \subseteq T_{mac} \tag{3.2}$$

dimana:
CFT: *Channel Free Time*
T_{busy} = Waktu saat media kanal sedang sibuk
T_{mac} = waktu penggunaan media nirkabel disekitar *node*.

Nilai dari perhitungan ini kemudian diteruskan ke lapisan jaringan untuk membantu mengambil keputusan rute terbaik pada proses *routing* khususnya untuk DSR pada tugas akhir ini. Dengan kata lain berarti lapisan jaringan akan mengambil data dari lapisan MAC. Pada NS-2, cara untuk menghubungkan kedua lapisan ini adalah dengan menuliskan list program sebagai berikut pada file mac-802_11.cc pada folder NS-2.

```
#include “mac/mac802_11.h”
```

Setelah itu, kita memodifikasi file mac802-11.cc pada folder yang sama dengan menambahkan list program sebagai berikut.

```
Void
Mac 802_11::recv(Packet *p, Handler *h)
{
...
   if(tx_active_ && hdr->error() == 0) {
```

```
      hdr->error () = 1;
   }

hdr_mac802_11 *mh = HDR_MAC802_11 (p);
u_int32_t idNode = ETHER_ADDR(mh->dh_ta);
double power = p->txinfo.RxPr; insertNode
(idNode,power,nodesPower);

...

}

struct time_power {
int pos;
double power[20];
u_int32_t idNode;


}
```

Jika terdapat kerusakan data atau gangguan yang menyebabkan pengiriman informasi gagal, maka MAC layer akan memberikan informasi kepada lapisan jaringan. Sesuai dengan referensi [9], maka penambahan list program berikut juga perlu ditambahkan dalam file Packet.h. Tulisan dengan huruf tebal adalah tambahan program yang dituliskan sesuai dengan referensi tersebut. Sedangkan program dengan huruf tidak tebal adalah program yang sudah ada pada perpustakaan NS-2. Hal ini untuk memperlihatkan posisi dari penambahan program tersebut.

```
(…)

int xmit_reason_;
#define XMIT_REASON_RTS 0X01
#define XMIT_REASON_ACK 0X02

// ktnahm add for DAMPEN policy
```

```
#define XMIT_REASON_CONFIRM 0X03

// Alonso – Rocha
#define XMIT_REASON_HiGH_POWER 0X04

(…)
```

Selain penambahan program pada file packet.h, file program dengan nama mac802-11.cc juga perlu ditambahkan. File ini berada pada folder NS-allinone-2.35. Penambahan program ini untuk memberikan notifikasi jika terjadi kesalahan link pada protokol routing DSR ini [9].

Lapisan MAC bekerja secara normal untuk mengirimkan informasi RTS (request to send) kepada *node* tetangganya. Namun jika komunikasi ini tidak berjalan, lapisan MAC harus menunggu dan mencoba lagi kemudian [9].

```
if (ch->xmit_failure_) {

struct hdr_mac802_11* dh = HDR_MAC802_11(pktTx_); uint32_t
idNode = ETHER_ADDR(dh->dh_ra);
double received_Power = pktTx_->txinfo_.RxPr;

int i = findNode(idNode, nodesPower);

if (i != -1)  received_Power = nodesPower[i].power[nodesPower[i].pos];

    if (received_Power > RxThreshold )
    {

ch->size() -= phymib_.getHdrLen11();
ch->xmit_reason_ = XMIT_REASON_HIGH_POWER;
ch->xmit_failure_(pktTx_->copy(),ch->xmit_failure_data_);
}
```

```
else {
ch->size() -= phymib_.getHdrLen11(); ch->xmit_reason_ =
XMIT_REASON_RTS;
ch->xmit_failure_(pktTx_->copy(),
ch->xmit_failure_data_);
}
```

3.3.4 DSR Menggunakan IEEE 802.11 Baru

Setelah proses penambahan CFT pada lapisan mac, protokol DSR akan berjalan sesuai dengan kerja dari lapisan mac yang baru tersebut. File *MAC layer* yang baru ini kemudian disimulasikan keenam skenario pasangan koneksi dari sistem VANET menggunakan protokol DSR 50 *node* seperti yang telah dilakukan sebelumnya kemudian hasil simulasinya akan dikorelasikan sebelum dan setelah penggunaan metode optimasi *cross layer* ini.

Untuk menggunakan file IEEE 802.11 baru ini, maka langkah yang harus dilakukan adalah dengan mengganti parameter simulasi sebagai berikut.

```
set opt(chan)       Channel/WirelessChannel    ;# channel type
set opt(prop)       Propagation/TwoRayGround   ;# radio-propagation
model
set opt(netif)      Phy/WirelessPhy            ;# network interface type
set opt(mac)        Mac/802_11New              ;# New MAC type
```

Halaman ini sengaja dikosongkan

BAB 4

PENGUJIAN DAN ANALISIS

4.1 Pendahuluan

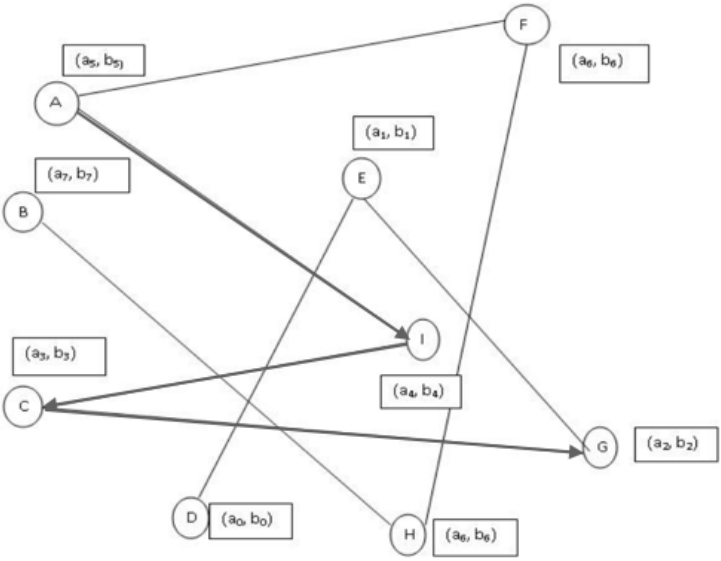
Pada bab ini dijelaskan analisa sistem berdasarkan data yang diperoleh dari simulasi dan pengujian sistem yang telah dilakukan pada bab 3 mengenai *dynamic source routing* pada VANET. Sistem dirancang dengan menggunakan parameter simulasi yang bekerja sesuai dengan karakterisik pada VANET. Sistem ini dijalankan menggunakan protokol routing DSR dengan 50 *node*. Setelah itu ditambahkan metode *cross layer* yaitu perhitungan *channel free time* pada *MAC layer* 802.11 perpustakaan NS-2. Hasil dari simulasi yang dilakukan kemudian dibandingkan nilai rata – rata end-to-end delay dan juga jumlah dari packet loss antar keenam skenario simulasi protokol routing DSR sebelum menggunakan metode optimasi cross layer. Barulah setelah itu membandingkan protokol routing DSR sebelum dan sesudah menggunakan metode optimasi *cross layer*.

4.2 Analisa Mobilitas *Node*

Analisa pertama adalah melihat pengaruh dari penggunaan *random mobility waypoint* pada simulasi yang akan dijalankan. *Random mobility waypoint* digunakan untuk mendapatkan mobilitas acak dari kendaraan atau *node* pada VANET. Pergerakan acak ini didapat dengan menggunakan bantuan program yang sudah ada pada perpustakaan NS-2. Penggunaan program ini ditujukan untuk simulasi DSR 50 *node*.

Gambar 4.1 dibawah ini merepresentasikan bagaimana random mobility waypoint ini bekerja. Pada random mobility waypoint, seluruh *node* bergerak bebas secara acak ke segala arah dalam batasan luas daerah simulasi tertentu. Pergerakan *node* ini diatur selama waktu simulasi berjalan sampai mencapai tujuan dari pergerakan tersebut. Ketika sudah sampai ditujuannya, maka *node* akan berhenti selama pause time lalu kemudian mengulang proses yang sama setelahnya. Pada tugas akhir ini, pause time ditetapkan 0 detik sehingga seluruh *node* terus bergerak secara kontinyu yang menyebabkan topologi jaringan yang selalu berubah.

Perubahan rute adalah ketika *node* A yang berada pada (a5, b5), bergerak kearah *node* I pada (a4, b4) kemudian pada rentang waktu tertentu *node* A kembali bergerak kearah yang berbeda *node* C (a3, b3). Perubahan pergerakan dari *node* A yang bergerak kearah *node* I lalu



Gambar 4.1 Ilustrasi Random Mobility Waypoint

mengubah arahnya ke *Node* C dihitung sebagai satu kali perubahan rute. Sedangkan perubahan link adalah ketika koneksi antar *node* sumber dan tujuan berubah karena pergerakan dari *node* tersebut.

Simulasi DSR 50 *node* menggunakan program *random mobility waypoint* dan juga CBR *generator* pada perpustakaan NS-2. Gambar 4.1 merepresentasikan perubahan rute dan perubahan link yang terjadi pada simulasi DSR 50 *node* ini. *Node* 0 sampai dengan *node* 49 mengalami perubahan rute yang jumlahnya cukup banyak. Jika dilihat dari tabel 4.1, dari seluruh *node* yang bergerak maka didapatkan total perubahan rute yang dilakukan oleh setiap *node* mencapai 10141 kali selama 800 detik waktu simulasi. Selain itu, total perubahan link mencapai 153 kali. Kedua angka ini mengindikasikan jika pause time diatur 0 detik, maka perubahan rute dan juga perubahan linknya bernilai besar karena mobilitas yang tinggi dari setiap *node*. Atas dasar ini maka perlunya optimasi untuk mengatasi proses pengiriman data dari satu *node* ke *node* lainnya mengingat *node* yang selalu bergerak secara acak sehingga membuat topologi jaringan berubah dan kepadatan kanal.

Tabel 4.1 Perubahan Rute dan Perubahan Link DSR 50 *Node*

Route Changes	Link Changes
10141	153

4.3 Performa Protokol DSR dengan 50 *Node* dalam VANET

Analisa yang dilakukan pada bagian ini adalah analisa terhadap simulasi *dynamic source routing* (DSR) pada VANET dengan menggunakan 50 *node*. Terdapat enam skenario yang seluruhnya akan dianalisa. Parameter yang dilihat adalah *end-to-end delay* dan juga *packet loss*.

4.3.1 Performa Sistem dengan 1 Koneksi

Analisa DSR ini menggunakan 50 *node* dengan satu *node* sumber. *Node* 1 dan *Node* 2 dihubungkan dengan koneksi TCP, dimana *node* 1 bertindak sebagai *node* sumber dan *node* 2 sebagai *node* tujuan. Skenario dari simulasi ini adalah *node* 1 berjalan mendekati *node* 2 untuk mengirimkan data, dimana pada saat proses *routing* pertama kali adalah *node* sumber mengirimkan paket *request* (RREQ) kepada *node* tetangganya. Simulasi ini dilakukan untuk melihat cara kerja dari protokol *dynamic source routing* itu sendiri. Hal lainnya yang dapat dilihat dari simulasi ini yaitu menjadi dasar besaran *delay* dan *packet loss* untuk dijadikan acuan hasil simulasi lainnya.

Tabel 4.2 dibawah ini adalah nilai informasi mengenai hasil simulasi yang telah dijalankan. Hasil end-to-end delay yang didapat masih perlu dikurangi karena semakin kecil *delay* maka akan semakin baik kinerja dari jaringan ini. Apalagi simulasi ini menggunakan 1 pasangan koneksi yang seharusnya nilai dari *end-to-end delay* ini dapat lebih kecil lagi mengingat belum banyaknya jumlah sumber yang digunakan. Selain itu jika diperhatikan rasio perbandingan dari jumlah paket yang dikirim dengan paket yang dibangkitkan adalah 0.98 atau 98% yang berarti hampir seluruh paket dapat dikirimkan. Akan tetapi terdapat 33 drop paket atau paket tersebut tidak dapat mencapai tujuan karena kepadatan kanal yang disebabkan oleh pengiriman data secara kontinyu pada topologi jaringan yang selalu berubah – ubah karena mobilitas dari tiap *node* yang tinggi.

Tabel 4.2 Informasi Simulasi DSR 50 *Node* Satu Koneksi

Simulation Information	
Number of Generated Packets	1330 *packets*
Number of Sent Packets	1306 *packets*
Number of Forwarded Packets	2204 *packets*
Number of Dropped Packets	33 *packets*
Number of Loss Packets	79 *packets*
Minimal Packet Size	32
Maximal Packet Size	608
Average Packet Size	443.2965
Number of Sent Bytes	459192 *bytes*
Number of Forwarded Bytes	1184884 *bytes*
Number of Dropped Bytes	6240 *bytes*
Minimal Delay	0.005579 sec
Maximal Delay	27.59935 sec
Average Delay	0.0667 *sec*

Jika dihitung rasio perbandingan antara *packet loss* dengan paket terkirim adalah 0.06 atau 6 %. Angka ini masih perlu dikurangi mengingat pentingnya data yang dikirimkan pada VANET untuk mencegah terjadinya kecelakaan antar kendaraan.

4.3.2 Performa Sistem dengan 2 Koneksi

Pada bagian ini, simulasi DSR dengan 50 *node* sudah mulai menggunakan program cbrgen.tcl untuk membuat koneksi antara *node* sumber dan *node* tujuannya. Koneksi yang digunakan pada lapisan transport juga sama yaitu UDP dengan trafik berupa CBR.

Jumlah paket yang dibangkitkan jauh lebih banyak dibandingkan dengan simulasi pada satu koneksi saja. Hal ini dapat terjadi karena jumlah dari *node* sumber yang lebih banyak sehingga jumlah dari *node* yang dibangkitkan juga bertambah. Jika dilihat dari jumlah packet loss, ada sebanyak 79 paket dengan rasio paket hilang adalah 0.0387 atau 3.87%. Angka ini justru menunjukkan pengurangan dari skenario sebelumnya. Hal ini terjadi karena walaupun mobilitas node bergerak secara kontinyu, namun seluruh node tujuan dapat dicapai sehingga tidak terlalu mempengaruhi jumlah packet lossnya.

Nilai average delay masih bisa ditoleransi karena berada dibawah 1 detik. Hanya saja, jika dilihat nilai maksimal delay dari simulasi ini sangat besar yaitu 27.93 detik. Hal ini berarti pada simulasi sempat terjadi delay tersebut yang menandakan kondisi jaringan yang belum optimal.

Tabel 4.3 Informasi Simulasi DSR 50 *Node* Dua Koneksi

Simulation Information	
Number of Generated Packets	2070 *packets*
Number of Sent Packets	2038 *packets*
Number of Forwarded Packets	2993 *packets*
Number of Dropped Packets	40 *packets*
Number of Loss Packets	79 *packets*
Minimal Packet Size	32
Maximal Packet Size	608
Average Packet Size	472.5022
Number of Sent Bytes	861204 *bytes*
Number of Forwarded Bytes	1601552 *bytes*
Number of Dropped Bytes	7092 *bytes*
Minimal Delay	0.005579 sec
Maximal Delay	27.9314 sec
Average Delay	0.04016 *sec*

4.3.3 Performa Sistem dengan 4 Koneksi

Bagian ini menjelaskan performa sistem dengan menggunakan 4 koneksi. Tabel 4.4 di bawah ini adalah nilai informasi mengenai hasil simulasi yang telah dijalankan. Hasil end to end delay yang didapat malah lebih kecil dari sistem dengan menggunakan 2 koneksi. Memang pergerakan node dibuat sama pada seluruh variasi, namun angka ini menunjukkan bahwa setiap node sumber pada skenario ini menggunakan jalur yang berbeda dengan node sumber pada skenario sebelumnya. Terdapat 43 drop paket atau paket tersebut tidak dapat mencapai tujuan karena kepadatan kanal yang disebabkan oleh pengiriman data secara kontinyu pada topologi jaringan yang selalu berubah – ubah karena mobilitas dari tiap *node* yang tinggi.

Tabel 4.4 Informasi Simulasi DSR 50 Node Empat Koneksi

Simulation Information	
Number of Generated Packets	3476 *packets*
Number of Sent Packets	3447 *packets*
Number of Forwarded Packets	4273 *packets*
Number of Dropped Packets	43 *packets*
Number of Loss Packets	106 *packets*
Minimal Packet Size	32
Maximal Packet Size	608
Average Packet Size	487.6757
Number of Sent Bytes	1580640 *bytes*
Number of Forwarded Bytes	2287196 *bytes*
Number of Dropped Bytes	10208 *bytes*
Minimal Delay	0.0055186 sec
Maximal Delay	27.54703 sec
Average Delay	0.0288 *sec*

4.3.4 Performa Sistem dengan 6 Koneksi

Di bawah ini adalah hasil simulasi dengan menambahkan variasi koneksi menjadi enam. Tabel 4.5 di bawah ini menunjukkan kelima koneksi antar *node* sumber dan tujuan, dimana terdapat empat *node* sumber saja. Jumlah paket yang dibangkitkan jauh lebih banyak dibandingkan dengan simulasi pada variasi koneksi sebelumnya. Hal ini dapat terjadi karena jumlah dari *node* sumber yang lebih banyak sehingga jumlah dari *node* yang dibangkitkan juga bertambah.

Sedangkan jika dilihat dari jumlah packet loss, ada sebanyak 126 paket dengan rasio paket hilang adalah 0.025 atau 2.5 %. Angka ini berkurang dibandingkan dengan simulasi sebelumnya. Berkurangnya nilai packet loss ini terjadi selain karena jalur yang digunakan tidak sepadat pada skenario simulasi dengan menggunakan empat koneksi.

Nilai average delay pada skenario ini cukup baik karena rentang dari minimum dan maksimum delay nya tidak terlalu jauh. Selain itu nilai dari delay ini juga lebih kecil dari skenario sebelumnya yang dapat terjadi karena penggunaan jalur yang tidak terlalu padat dibandingkan dengan simulasi sebelumnya.

Tabel 4.5 Informasi Simulasi DSR 50 *Node* Enam Koneksi

Simulation Information	
Number of Generated Packets	5114 *packets*
Number of Sent Packets	5094 *packets*
Number of Forwarded Packets	5857 *packets*
Number of Dropped Packets	47 *packets*
Number of Loss Packets	126 *packets*
Minimal Packet Size	32
Maximal Packet Size	620
Average Packet Size	471.008
Number of Sent Bytes	2370936 *bytes*
Number of Forwarded Bytes	3052652 *bytes*
Number of Dropped Bytes	13696 *bytes*
Minimal Delay	0.0054977 sec
Maximal Delay	1.487072 sec
Average Delay	0.01707 *sec*

4.3.5 Performa Sistem dengan 8 Koneksi

Sesuai dengan referensi [1], penulis melakukan kajian terhadap protokol DSR dalam VANET dengan menghubungkan TCP/UDP sebanyak 8 koneksi. Terdapat 8 hubungan antara *node* sumber dengan *node* tujuan, dimana jumlah *node* sumbernya adalah 7. Seluruh koneksi ini juga dibuat dengan bantuan program cbrgen.tcl yang terdapat pada perpustakaan NS-2.

Tabel 4.6 Data *Node* Sumber dan Tujuan Simulasi 8 Koneksi

Koneksi	*Node* Sumber	*Node* Tujuan
1	1	2
2	2	3
3	2	4
4	9	10
5	12	13
6	13	14
7	16	17
8	18	19

Pasangan koneksi ini dapat dilihat pada tabel 4.6 diatas. Perlu diketahui juga setiap *node* sumber dapat menjadi *node* tujuan begitupun sebaliknya. Hal ini juga sesuai dengan kondisi nyata yaitu setiap kendaraan di jalan raya tentunya akan mengirimkan data dan juga menerima data dari kendaraan lain.

Tabel 4.7 di bawah merupakan tabel berupa informasi simulasi sistem dengan skenario menggunakan 8 koneksi. Jumlah paket yang dibangkitkan bertambah dibandingkan dengan kedua simulasi sebelumnya karena memang jumlah *node* sumbernya juga bertambah. Adapun rasio packet loss terhadap paket yang dikirimkan adalah 0.0276 atau 2.76 %. Nilai ini bertambah terhadap performa sistem dengan 6 koneksi. Hal ini mungkin terjadi karena kondisi jaringan tidak sama padatnya dengan simulasi sebelumnya. Namun, jika diperhatikan memang jumlah packet loss lebih banyak dari simulasi sebelumnya.

Tabel 4.7 Informasi Simulasi DSR 50 *Node* 8 Koneksi

Simulation Information	
Number of Generated Packets	6692 *packets*
Number of Sent Packets	6669 *packets*
Number of Forwarded Packets	7108 *packets*
Number of Dropped Packets	62 *packets*
Number of Loss Packets	184 *packets*
Minimal Packet Size	32
Maximal Packet Size	644
Average Packet Size	482.5444
Number of Sent Bytes	3134012 *bytes*
Number of Forwarded Bytes	3753700 *bytes*
Number of Dropped Bytes	21876 *bytes*
Minimal Delay	0.0054973 sec
Maximal Delay	7.0907 sec
Average Delay	0.01874 *sec*

Begitupun dengan nilai delay, dimana simulasi ini memiliki delay yang lebih besar karena alasan yang sama yaitu kepadatan kanal tidak sama padatnya dengan simulasi sebelum ini. Bertambahnya koneksi membuat kondisi jaringan ini lebih padat dan jika dilihat letak dari *node* sumber satu dan lainnya juga membuat jaringan ini menjadi lebih padat.

Alasan utama adalah jika antar *node* sumber yang satu dengan lainnya diletakkan berjauhan, maka kanal yang digunakan akan berbeda sehingga dapat mengurangi atau menambah kongesi pada jaringan.

4.3.6 Performa Sistem dengan 10 Koneksi

Simulasi ini dilakukan dengan menambahkan koneksi menjadi 10. Dengan demikian akan terdapat 10 hubungan TCP/UDP antara *node* sumber dengan *node* tujuan, dimana terdapat 9 *node* sumber. 8 koneksi pertama sama dengan simulasi sebelumnya dengan penambahan dua koneksi yaitu *node* 19 sebagai *node* sumber dihubungkan *node* 20 dan juga *node* 22 sebagai *node* sumber yang dihubungkan dengan tujuannya yaitu *node* 23.

Tabel 4.8 Informasi Simulasi DSR 50 *Node* 10 Koneksi

Simulation Information	
Number of Generated Packets	8612 *packets*
Number of Sent Packets	8576 *packets*
Number of Forwarded Packets	12464 *packets*
Number of Dropped Packets	97 *packets*
Number of Loss Packets	299 *packets*
Minimal Packet Size	32
Maximal Packet Size	728
Average Packet Size	476.3823
Number of Sent Bytes	3922724 *bytes*
Number of Forwarded Bytes	9940182 *bytes*
Number of Dropped Bytes	6533104 *bytes*
Minimal Delay	0.00549759 sec
Maximal Delay	80.44526 sec
Average Delay	0.047383 *sec*

Simulasi ini juga membuat packet loss bertambah dengan rasio packet loss terhadap paket terkirim sebesar 0.034 atau 3.4 persen. Angka ini bertambah terhadap simulasi sistem dengan 8 koneksi. Bertambahnya persentase packet loss ini disebabkan karena beberapa hal. Alasan pertama yaitu memang dari setiap simulasi yang dijalankan pergerakan *node* pada jaringan ini diatur tetap, namun tetap saja pergerakan ini

membuat topologi jaringan berubah sehingga mempengaruhi proses pengiriman paket informasi. Alasan kedua walaupun dengan 10 koneksi menghasilkan rasio packet loss yang lebih besar daripada simulasi sebelumnya, ditambah lagi posisi dari *node* relay yang meneruskan paket juga perlu dipertimbangkan. Jika *node* relay tersebut berada pada jalur dengan kondisi padat, maka paket tersebut tidak dapat dikirim dan merupakan penyebab suatu paket dapat hilang. Perlu diperhatikan juga bahwa jumlah paket drop bertambah menjadi 97 paket seiring bertambahnya packets dropping *node*.

Ditinjau dari delay simulasinya, maka terjadi penambahan delay menjadi 0.0473 detik. Hal ini mengindikasikan kepadatan kanal juga terjadi pada jaringan ini. Mobilitas *node* memang dibuat sama namun dengan penambahan *node* sumber menjadi 9 membuat sistem ini dapat menggunakan kanal lainnya dengan kondisi yang tidak padat.

4.3.7 Performa Sistem dengan 12 Koneksi

Sistem ini menggunakan 12 koneksi dengan 10 koneksi pertama sama dengan simulasi sebelumnya. Terdapat penambahan dua koneksi yaitu *node* 23 yang dihubungkan dengan *node* 24 dan *node* 27 yang dihubungkan dengan *node* 28. *Node* 23 dan *node* 28 bertindak sebagai *node* sumber sedangkan *node* 24 dan *node* 27 bertindak sebagai *node* tujuan. Jika dibandingkan dengan simulasi sebelumnya, maka pada simulasi ini menunjukkan hasil yang tidak terlalu jauh dengan simulasi sebelumnya.

Di bawah ini adalah tabel hasil simulasi sistem VANET dengan menambahkan koneksinya menjadi 12 koneksi. Rasio packet loss terhadap paket yang terkirim adalah 0.04 atau 4 %. Rasio ini juga tidak terlampau terlalu jauh dengan simulasi – simulasi sebelumnya walaupun memang angka ini bertambah dengan sistem menggunakan 10 koneksi.

Sebaliknya, nilai rata – rata delay justru berkurang menjadi 0.03349 detik. Terjadi pengurangan sebesar 0.013893 detik atau sebesar dua persen. Hal ini berarti kondisi jaringan dan tingkat kepadatannya tidak jauh berbeda dengan hasil simulasi sebelumnya yaitu simulasi dengan menggunakan 10 koneksi.

Tabel 4.9 Informasi Simulasi DSR 50 *Node* 12 Koneksi

Simulation Information	
Number of Generated Packets	10619 *packets*
Number of Sent Packets	10574 *packets*
Number of Forwarded Packets	17797 *packets*
Number of Dropped Packets	106 *packets*
Number of Loss Packets	433 *packets*
Minimal Packet Size	32
Maximal Packet Size	672
Average Packet Size	480.9679
Number of Sent Bytes	4757618 *bytes*
Number of Forwarded Bytes	9401820 *bytes*
Number of Dropped Bytes	39084 *bytes*
Minimal Delay	0.0054586 sec
Maximal Delay	27.73686 sec
Average Delay	0.03349 *sec*

4.3.8 Performa Sistem dengan 14 Koneksi

Sistem dengan 14 koneksi juga menggunakan koneksi – koneksi sebelumnya antara hubungan *node* sumber dengan *node* tujuan. 12 koneksi pertama menggunakan koneksi pada simulasi sebelumnya dengan penambahan dua koneksi lagi. Jadi 14 koneksi ini menggunakan seluruh koneksi dalam tugas akhir ini seperti yang terlihat pada tabel 4.8.

Sedangkan untuk hasil simulasinya. Jumlah paket yang dibangkitkan juga lebih banyak dengan jumlah packet loss pada simulasi ini juga bertambah. Akan tetapi jika dihitung rasio packet loss pada simulasi ini adalah 0.039 atau 3.9 %. Jadi, tidak terpaut jauh antara simulasi sebelumnya. Namun, rasio ini juga tidak menunjukkan kondisi yang ideal karena memang seharusnya paket – paket tersebut harus seluruhnya terkirim. Namun jumlah paket drop yang lebih kecil dibandingkan dengan packet loss menandakan bahwa masalah utama dari kondisi jaringan ini adalah dari lamanya waktu tunggu dari suatu paket yang sedang dikirimkan. Karena lamanya waktu tunggu tersebut, maka paket – paket akan didrop dan tidak dapat dikirimkan. Sistem ini menandakan kondisi jaringan VANET dengan 13 *node* sumber masih dapat bekerja lebih baik lagi karena jumlah packet loss harus ditekan seminimal mungkin. Tingkat kepadatan jaringan karena proses

pengiriman informasi yang begitu tinggi ditambah mobilitas *node* yang menyebabkan topologi jaringan berubah sehingga jumlah dari paket yang hilang ini juga besar.

Tabel 4.10 Data *Node* Sumber dan Tujuan Simulasi dengan 14 Koneksi

Koneksi	*Node* Sumber	*Node* Tujuan
1	1	2
2	2	3
3	2	4
4	9	10
5	12	13
6	13	14
7	16	17
8	18	19
9	19	20
10	22	23
11	23	24
12	27	28
13	27	29
14	28	29

Hasil delay justru menunjukkan peningkatan menjadi 0.0287 detik. Pada skenario ini justru memiliki delay yang lebih kecil dibandingkan simulasi sebelumnya. Untuk itu kinerja dari jaringan VANET ini perlu dioptimasi. Karakteristik dari VANET memang membuat nilai parameter - parameter ini menjadi besar. Terlebih lagi terdapat pergerakan *node* yang dari diam seketika bergerak menjauhi atau mendekati dari *node* sumber. Begitupun dengan pergerakan *node* relay jika menjauhi dari *node* sumber, maka paket kemungkinan akan didrop. Seluruh 13 *node* sumber ini bekerja tidak dalam waktu yang bersamaan, menyesuaikan kondisi nyata pada jalan raya dimana mobil tidak selalu mengirimkan data seperti saat sedang parkir ataupun dalam kemacetan.

Jika *node* tersebut menjadi *node* tujuan untuk dua *node* sumber yang berbeda, maka hal ini juga akan berdampak terhadap kondisi kepadatan jaringannya. Karena paket informasi yang datang dalam waktu bersamaan, salah satu paket harus menunggu dalam rentang waktu tertentu sehingga menyebabkan kepadatan jaringan.

Tabel 4.11 Informasi Simulasi DSR 50 *Node* 14 Koneksi

Simulation Information	
Number of Generated Packets	12584 *packets*
Number of Sent Packets	12533 *packets*
Number of Forwarded Packets	20328 *packets*
Number of Dropped Packets	121 *packets*
Number of Loss Packets	489 *packets*
Minimal Packet Size	32
Maximal Packet Size	652
Average Packet Size	478.713
Number of Sent Bytes	5612336 *bytes*
Number of Forwarded Bytes	10759556 *bytes*
Number of Dropped Bytes	45392 *bytes*
Minimal Delay	0.05496773 sec
Maximal Delay	1.51708 sec
Average Delay	0.02587 *sec*

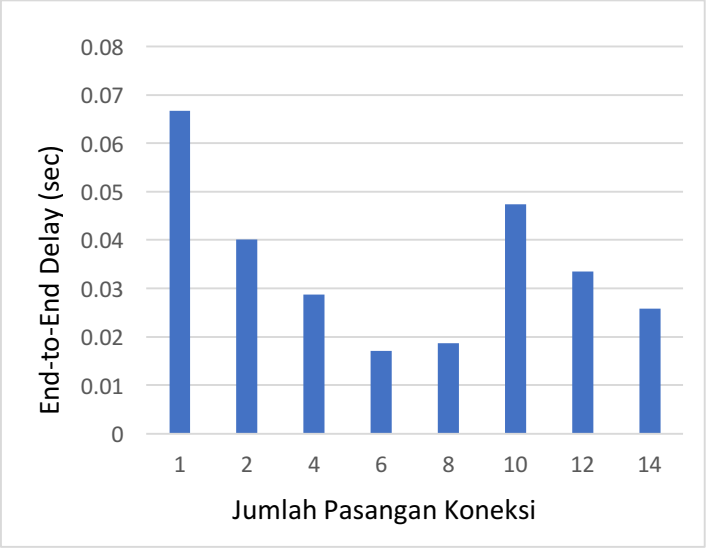
4.4 Perbandingan Seluruh Variasi Koneksi pada Sistem

Dari seluruh skenario simulasi protokol routing DSR pada VANET ini, dapat dibandingkan satu sama lainnya nilai dari end-to-end delay dan juga packet loss ratio. Secara umum seluruh skenario ini menunjukkan performa yang cukup baik dari segi delay, namun jika dilihat dari persentase packet lossnya masih berada dikisaran 3 – 5 %.

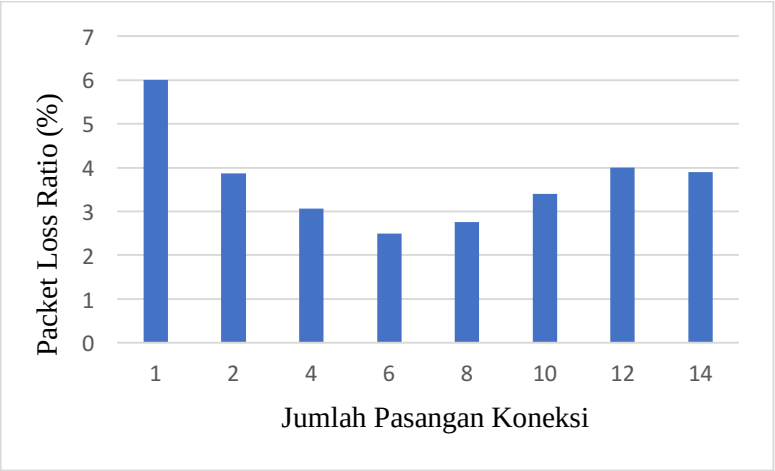
Gambar 4.2 dibawah adalah grafik rata- rata delay dari seluruh simulasi yang dijalankan. Terjadi lonjakan di awal pasangan koneksi saat simulasi menggunakan satu koneksi dengan delay mencapai 0.067 detik atau 6.7 milidetik. Selain itu, terlihat juga bertambahnya jumlah koneksi ternyata tidak selalu membuat delay nya juga bertambah. Hal ini terlihat saat simulasi dengan empat koneksi memiliki delay yang lebih kecil daripada simulasi dengan satu dan dua koneksi. Hal ini dapat terjadi karena letak dari *node – node* sumber dan tujuannya. Jika pada empat koneksi *node* sumber mengirimkan data kepada *node* tujuannya melalui salah satu lintasan, maka pada simulasi menggunakan satu ataupun dua koneksi mungkin pertukaran data menggunakan lintasan yang tidak terlalu padat. Hal ini dapat berdampak pada nilai delay dan juga packet

loss. Namun jika dilihat, kinerja dari jaringan ini cukup baik dari sisi delaynya karena kalau dihitung rata – rata keseluruhannya bernilai 0.0347 detik. Angka ini menunjukkan bahwa kinerja jaringan sudah baik karena masih berada di bawah nilai 0.05 detik namun masih harus diperbaiki dan harus ditekan seminimum mungkin mengingat pentingnya pertukaran data pada jaringan VANET.

Jika dilihat dari persentase packet loss, juga terjadi lonjakan yang cukup signifikan pada penggunaan satu koneksi lalu turun kembali ke kisaran nilai 2.5 - 3 %. Perlu diketahui semakin besar jumlah koneksinya maka semakin besar juga paket yang dibangkitkan. Selain itu jika dilihat pada tabel – tabel hasil simulasi, bertambahnya koneksi ini juga meningkatkan jumlah packet loss. Hanya saja, persentase packet loss dengan paket terkirimnya tidak selalu naik. Artinya, mobilitas *node* dan protokol routing ini bekerja cukup baik karena dapat menjaga persentase packet loss ini. Hal inilah yang perlu dioptimasi karena jika pada saat pertukaran data di jalan raya ini tidak sampai atau bahkan data tersebut hilang, maka tujuan dari VANET untuk mengurangi kecelakaan justru tidak dapat tercapai.



Gambar 4.2 Kinerja Delay dari Keenam Skenario Simulasi



Gambar 4.3 Rasio Packet Loss dari Seluruh Skenario Simulasi

4.5 Analisa Penambahan Metode Optimasi Cross Layer

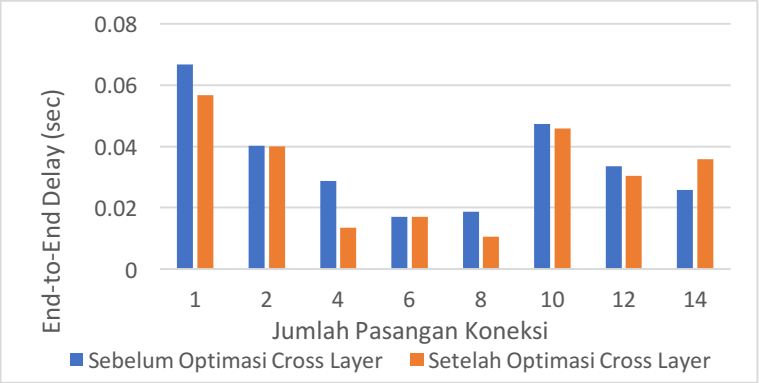
Metode optimasi *cross layer* ini ditambahkan ke *file* mac 802.11 yang berada pada perpustakaan NS-2. *List* program yang sudah ada ditambahkan perhitungan *channel free time* (CFT) yang hasilnya digunakan pada *network layer* saat proses *routing* berjalan. Ketika *node* sumber mengirimkan paket *request* (RREQ), maka hasil dari perhitungan CFT ini akan mencegah pengiriman paket kedalam kanal yang kondisinya sedang padat. Karena pencegahan ini jumlah *packet loss* dapat berkurang dan juga dapat mengurangi nilai *end-to-end delay*.

Jika dikaitkan dengan teori pada referensi – referesi yang ada, hasil yang diharapkan dengan penambahan algoritma ini dapat mengurangi nilai *end-to-delay* sampai dengan 0.5 detik. Namun pengurangan ini memang karena pada referensi terkait, delay yang didapat dari setiap pasang koneksi memang lebih besar dibandingkan dengan nilai delay pada tugas akhir ini yaitu berada pada kisaran 0.1 – 1 detik. Selain itu jumlah paket yang hilang juga dapat dikurangi karena memang metode ini tidak akan mengirimkan paket kedalam kanal yang sedang sibuk sehingga paket – paket yang akan dikirimkan berada pada kondisi menunggu (*buffer*).

Tabel 4.12 menunjukkan perbandingan kinerja DSR pada VANET sebelum dan sesudah penerapan metode cross layer ini dilihat dari nilai end-to-end delay dan serta terdapat perhitungan selisih antar keduanya dan rata – rata berikut standar deviasinya. Penerapan metode cross layer ini belum berdampak signifikan terhadap nilai end-to-end delay. Namun, terdapat tiga skenario dengan penurunan delay yang cukup baik yaitu pada skenario simulasi dengan pasangan koneksi berjumlah 1, 4, dan 8. Rata – rata penurunan end-to-end delay sesudah metode optimasi cross layer ini adalah 0.01247 detik dengan standar deviasi 0.02063 detik. Angka ini sudah cukup baik walaupun terdapat kenaikan end-to-end delay pada saat 14 pasangan koneksi. Selain dari ketiga pasangan koneksi ini, penurunan delay tidak terlalu signifikan dan justru memiliki nilai yang tidak terlalu jauh sistem sebelum ditambahkan metode optimasi cross layer. Hal ini dapat terjadi karena beberapa hal, salah satunya adalah sesuai referensi[6] mengatakan bahwa metode ini digunakan untuk protokol routing AODV. Selain itu kondisi dari jaringan yang lebih padat juga mempengaruhi hasil dari metode ini seperti yang terlihat pada pasangan koneksi berjumlah 6 dengan tanpa pengurangan dan 14 yang justru terjadi kenaikan end-to-end delay.

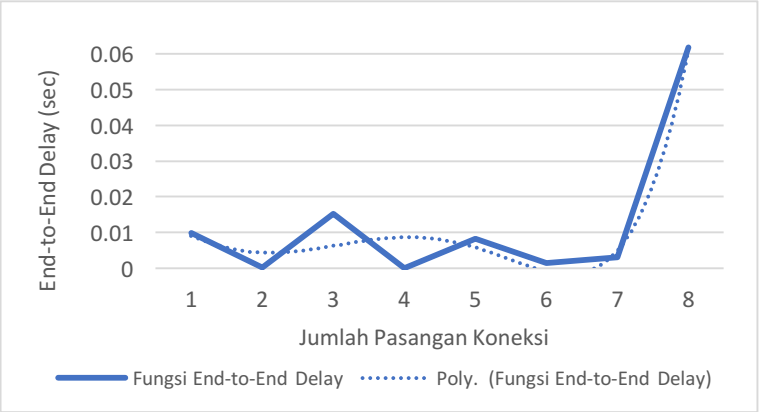
Tabel 4.12 Perbandingan Hasil End-to-End Delay Sebelum dan Sesudah Penerapan Optimasi Cross Layer

Comparison of Simulation Information				
Pairs	*End-to-End Delay Before Cross Layer Optimization*	*End-to-End Delay After Cross Layer Optimization*	*Errors*	*% Errors*
1	0.0667 *sec*	0.0567 *sec*	0.009 sec	0.9
2	0.04016 *sec*	0.04 sec	0.00016 sec	0.016
4	0.0288 *sec*	0.01357 sec	0.01523 sec	1.52
6	0.01707 *sec*	0.01707 sec	0 sec	0
8	0.01874 *sec*	0.0105 sec	0.00824 sec	0.8
10	0.047383 *sec*	0.04589 sec	0.001493 sec	0.14
12	0.03349 sec	0.0305 sec	0.00299 sec	0.29
14	0.02587 sec	0.0359 sec	+0.06177 sec	+6.1
		Rata - rata	0.012473 sec	1.22
		Standar deviasi	0.02063 sec	2



Gambar 4.4 Perbandingan End-to-End Delay Sebelum dan Sesudah Penerapan Optimasi Cross Layer

Gambar 4.4 adalah representasi berupa diagram batang yang menunjukkan perbandingan end-to-end delay sebelum dan sesudah penerapan optimasi cross layer pada sistem. Sedangkan gambar 4.5 adalah representasi berupa *trendline* yang berisikan fungsi matematis dimana perhitungan ini menghasilkan fungsi orde 5, yaitu : $y = 6.10^{-5}x^5 - 0.0009x^4 + 0.0041x^3 - 0.004x^2 - 0.0094x + 0.0194$. Fungsi orde 5 ini didapatkan karena data yang didapatkan dari hasil simulasi bervariasi.

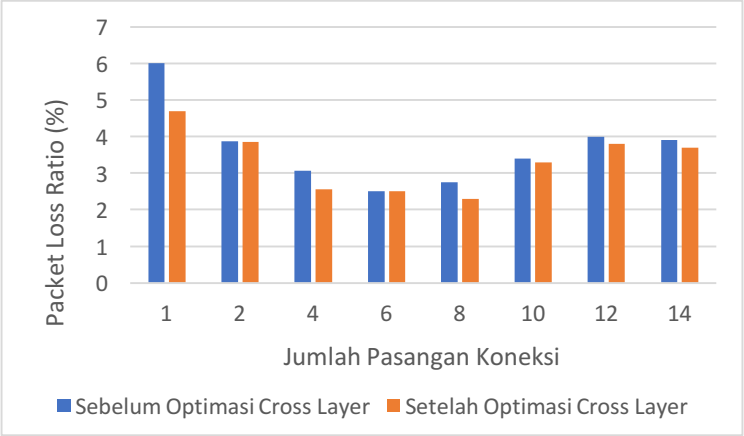


Gambar 4.5 *Trendline* End-to-End Delay Sesudah Penggunaan Optimasi Cross Layer

Tabel 4.13 menunjukkan perbandingan kinerja DSR pada VANET sebelum dan sesudah penerapan metode cross layer ini dilihat dari rasio rasio packet loss serta terdapat perhitungan selisih antar keduanya dan rata – rata berikut standar deviasinya. Seperti halnya dengan parameter end-to-end delay, Terdapat tiga skenario dengan penurunan rasio packet loss yang cukup baik yaitu pada skenario simulasi dengan pasangan koneksi berjumlah 1, 4, dan 8. Pada jumlah pasangan 1 koneksi merupakan pengurangan paling baik yaitu berjumlah 1.3%. Pada 4 pasangan koneksi pengurangan berjumlah 0.51% dan pada 8 pasang koneksi pengurangan sebesar 0.46%. Jika dilihat memang pada pasangan 1 koneksi ini memiliki pengurangan paling baik karena pada simulasi ini node sumber hanya memerlukan lebih sedikit jalur dibandingkan dengan jumlah pasangan koneksi lainnya. Jika pada pasangan koneksi berjumlah dua justru memiliki pengurangan paling sedikit, mungkin hal ini terjadi karena jalur yang dipakai sedang dalam kondisi padat dan memang nilai Cs yang didapat network layer dari mac layer tidak terlalu memberikan kontribusi signifikan pada proses routing yang bekerja.

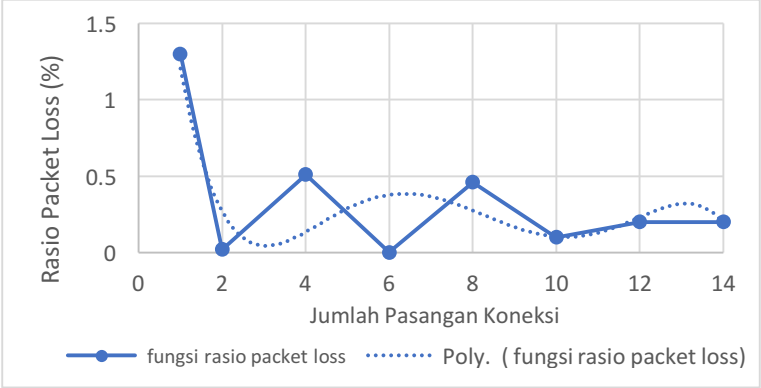
Tabel 4.13 Perbandingan Hasil Packet Loss Ratio Sebelum dan Sesudah Penerapan Optimasi Cross Layer

Comparison of Simulation Information			
Pairs	*Packet Loss Before Cross Layer Optimization (%)*	*Packet Loss After Cross Layer Optimization (%)*	*Errors*
1	6	4.7	1.3
2	3.87	3.85	0.02
4	3.07	2.56	0.51
6	2.5	2.5	0
8	2.76	2.3	0.46
10	3.4	3.3	0.1
12	4	3.8	0.2
14	3.9	3.7	0.2
		Rata - rata	0.35
		Standar deviasi	0.42



Gambar 4.6 Perbandingan Packet Loss Ratio Sebelum dan Sesudah Penerapan Optimasi Cross Layer

Gambar 4.6 merupakan representasi berupa diagram batang yang menunjukkan perbandingan rasio packet loss sebelum dan sesudah penerapan optimasi cross layer pada sistem dan gambar 4.7 adalah representasi berupa *trendline* tentang fungsi yang didapatkan juga menyerupai fungsi end-to-end delay yaitu orde 5: $y = -0.0002x^5 + 0.0098x^4 - 0.1485x^3 + 1.0239x^2 - 3.1066x + 3.4262$.



Gambar 4.7 *Trendline* Selisih Rasio Packet Loss Sesudah Penerapan Optimasi Cross Layer

Halaman ini sengaja dikosongkan

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil penelitian yang telah dilakukan maka dapat ditarik beberapa kesimpulan, diantaranya:

1. Metode Optimasi Cross Layer pada Tugas Akhir ini dapat mengurangi end-to-end delay pada skenario jumlah pasangan koneksi 1, 4, dan 8 dengan pengurangan sebesar 0.01 detik, 0.015 detik, dan 0.00824 detik.
2. Metode Optimasi Cross Layer pada Tugas Akhir dapat mengurangi nilai persentase packet loss sebesar 1.3%, 0.53%, dan 0.46% pada skenario jumlah pasangan koneksi 1,4, dan 8.
3. Bertambahnya jumlah koneksi pada sistem VANET dan mengatur parameter lainnya bernilai tetap tidak selalu membuat nilai end-to-end delay dan packet loss bertambah juga.
4. Rata – rata selisih perbedaan end-to-end delay sebelum dan sesudah penggunaan cross layer adalah 0.012473 detik.
5. Rata – rata selisih perbedaan rasio packet loss sebelum dan sesudah penggunaan cross layer adalah 0.35%.
6. Mobilitas dari tiap *node* dapat diatur dengan menggunakan random mobility waypoint dan mengatur pause time, dimana pause time dengan nilai 0 detik membuat mobilitas *node* lebih tinggi.
7. Besarnya nilai end-to-end delay dan packet loss dipengaruhi oleh mobilitas *node*, kepadatan jaringan, dan topologi jaringan yang selalu berubah.

5.2 Saran

Saran untuk penelitian selanjutnya adalah sebagai berikut:

1. Menjalankan simulasi VANET ini dengan menambahkan metode optimasi cross layer pada aplikasi network simulator generasi terbaru (NS-3) atau dengan simulator lainnya seperti NcTUns.
2. Penerapan metode optimasi cross layer dengan memperhatikan program yang sudah ada dari referensi yang terdapat pada tugas akhir ini.

3. Menjalankan simulasi dengan variasi jumlah koneksi dalam skala yang lebih besar.
4. Menambahkan kondisi lain sebagai variabel untuk melihat parameter selain dari sisi end-to-end delay dan packet loss.

DAFTAR PUSTAKA

[1] V. Ratwani, A. Shah, "Network Coding and Cross-Layer Approach for Reliability and Optimization of Routing in VANET: A Survey", International Journal of Computer Applications Volume 113 No.13, 2015.

[2] Armstrong, Lee, "Dedicated Short Range Communications (DSRC)",<https://en.wikipedia.org/wiki/Vehicular_communication_systems#cite_note-traffic_rep-1>, 2012

[3] Baumann, Rainer, "Vehicular Ad hoc Networks (VANET)", Master's Thesis in Computer Science ETH Zurich, 2004

[4] Johnson, David B, et al, "DSR: The *Dynamic source routing* Protocol for Multi-Hop Wireless Ad Hoc Networks", 2001

[5] D. B Jagannadha Rao, K. Sreenu, K. Karnam, "A Study on Dynamic Source Routing Protocol for Wireless Ad Hoc Networks", International Journal of Advanced Research in Computer and Communication Engineering, 2012

[6] P. Wannawilai, C. Sathitwiriyawong, "AODV with Sufficient Intermediary Bandwidth Aware", International Confrence On Electrical Engineering/Electronics Computer Telecommunications and Information Technology, 2010

[7] N. Yawan, P. Keeratiwintakorn, "AODV Improvement for Vehicular Networks with Cross Layer Technique and Mobility Prediction", International Symposium on Intelligent Signal Processing and Communication Systems, 2011

[8] K. Kamini & Rakesh, "VANET Parameters and Applications : A Review", Global Journal of Computer Science and Technology, 2010.

[9] Alonso Pequeno, Guillermo., Rocha Rivera, Javier, “Extension to MAC802.11 for Performance Improvement in MANET”, Karlstads Universitet, 2007.

[10] Mustafa, Ali Hassoune, et al. “Cross-Layer Solutions for Enhancing Multimedia Communication QoS Over Vehicular Ad hoc Networks”, The Sixth International Conference on Advances in Future Internet, 2014.

[11] Jarupan, Boangoat., Ekici, Eylem,“A Survey of Cross-Layer Design for VANETs”, 2010

[12] Pomplun, Robin., Datta, Amittava, “A Study of Long Distance Traffic Using the AODV Protocol in a Vehicular Ad Hoc Network” 66th IEEE Vehicular Technology Conference, 2007.

[13] Abedi, Omit., Fathy, Mahmood., Taghiloo, Jamshid, 2008 “Enhancing AODV Routing Protocol Using Mobility Parameters in VANET” IEEE/ACS Internasional Conference on Computer Systems and Applications, 2008

LAMPIRAN A

LEMBAR PENGESAHAN PROPOSAL TA

Departemen Teknik Elektro
Fakultas Teknologi Elektro – ITS

TE 141599 TUGAS AKHIR – 4 SKS

Nama Mahasiswa	:	Kevianda Kamarullah
Nomer Pokok	:	2213 100 174
Bidang Studi	:	Telekomunikasi Multimedia
Tugas Diberikan	:	Semester Genap 2016/2017
Dosen Pembimbing	:	1. Dr. Ir. Endroyono, DEA
		2. Dr. Ir. Wirawan, DEA
Judul Tugas Akhir	:	**Optimasi *Cross Layer* untuk Protokol *Dynamic Source Routing* Pada Komunikasi Antar Kendaraan Berbasis *Vehicular Ad-Hoc Networks* (VANETs)** **(*Cross Layer Optimization for Dynamic Source Routing Protocol on Vehicular Ad-Hoc Networks*)**

10 FEB 2017

Uraian Tugas Akhir :
Konsep *vehicle-to-vehicle communication* (V2V) pada *Intelligent Transportation Systems* (ITS) merupakan suatu konsep teknologi dimana antar kendaraan dapat berinteraksi satu sama lainnya. Salah satu komunikasi yang dapat menghubungkan V2V ini yaitu *Vehicular Ad-Hoc Networks* (VANETs). Pada VANET, mobil bertindak sebagai *node* dan terus bergerak sehingga membuat topologi jaringan berubah. Perubahan topologi jaringan ini mengakibatkan pengiriman data yang tidak maksimal seperti *delay* yang besar ataupun kegagalan dalam pengiriman data. Untuk itu upaya optimasi routing pada VANET ini perlu dikembangkan.
Tugas akhir ini membahas optimasi *cross layer* untuk *dynamic source routing* (DSR) pada VANET dengan mengintegrasikan *network layer dan MAC layer*. Pada *MAC layer* dilakukan proses kepadatan trafik untuk mengetahui ketersediaan kanal menggunakan *channel availability calculation* dan mengetahui perubahan jarak antar kendaraan menggunakan *mobility prediction method*. Hasil dari kedua metode ini diteruskan ke *network layer* untuk menentukan rute terbaik. Optimasi dilakukan dengan melakukan simulasi menggunakan NS-3 yang kemudian divalidasi antara perbandingan routing tipe DSR pada VANET sebelum dan sesudah menggunakan optimasi *cross layer*. Dengan optimasi *cross layer* ini diharapkan dapat mengurangi nilai *end-to-end delay* dan *routing overhead*.

Dosen Pembimbing I,

Dr. Ir. Endroyono, DEA
Nip : 196504041991021001

Dosen Pembimbing II,

Dr. Ir. Wirawan, DEA
Nip : 196311091989031011

Mengetahui,
Ketua Program Studi S1

I Dedet C. Riawan, ST. M.Eng. Ph. D.
Nip : 197311192000031001

Menyetujui,
Kepala Laboratorium Komunikasi Multimedia

Dr. Ir. Endroyono, DEA
Nip : 196504041991021001

Halaman ini sengaja dikosongkan

LAMPIRAN B

LISTING PROGRAM

1. Simulasi *Dynamic source routing* Menggunakan 50 *Node*

```
Antenna/OmniAntenna set Gt_ 1                ;#Transmit antenna gain
Antenna/OmniAntenna set Gr_ 1                ;#Receive antenna gain
Phy/WirelessPhy set L_ 1.0                   ;#System Loss Factor
Phy/WirelessPhy set freq_ 9.14e+08           ;#channel
Phy/WirelessPhy set bandwidth_ 11Mb          ;#Data Rate
Phy/WirelessPhy set Pt_ 0.281838             ;#Transmit Power
Phy/WirelessPhy set RXThresh_ 1.76149        ;#Receive Power Threshold
Mac/802_11 set dataRate_ 1Mb                 ;#Rate for Data    Frames
Mac/802_11 set basicRate_ 1Mb                ;#Rate for Control Frames

set opt(chan)         Channel/WirelessChannel    ;# channel type
set opt(prop)         Propagation/TwoRayGround   ;# propagation model
set opt(netif)        Phy/WirelessPhy            ;# network interface type
set opt(mac)          Mac/802_11                 ;# MAC type
set opt(ifq)          Queue/DropTail/PriQueue    ;# interface queue type
set opt(ll)           LL                         ;# link layer type
set opt(ant)          Antenna/OmniAntenna        ;# antenna model
set opt(ifqlen)       50                         ;# max packet in ifq
set opt(nn)           50                         ;# number of mobile*node*s
set opt(rp)           DSR                        ;# routing protocol
set opt(x)            1500                ;# X dimension of topography
set opt(y)            300                 ;# Y dimension of topography
set opt(stop)         800                 ;
set ns_           [new Simulator]

#Creating trace file and nam file
set tracefd       [open dsr50*node*.tr w]
set namtrace      [open dsr50*node*.nam w]
if { $opt(rp) == "DSR" } {
set opt(ifq)          CMUPriQueue
} else {
set opt(ifq)          Queue/DropTail/PriQueue
}
```

```
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)

# set up topography object
set topo       [new Topography]

$topo load_flatgrid $opt(x) $opt(y)

create-god $opt(nn)
# configure the node*s
      $ns_ node-config -adhocRouting $opt(rp) \
                -llType $opt(ll) \
                -macType $opt(mac) \
                -ifqType $opt(ifq) \
                -ifqLen $opt(ifqlen) \
                -antType $opt(ant) \
                -propType $opt(prop) \
                -phyType $opt(netif) \
                -channelType $opt(chan) \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTrace ON \
                -macTrace OFF \
                -movementTrace ON

    for {set i 0} {$i < $opt(nn) } { incr i } {
        set node_($i) [$ns_ node]
    }

source dsr50*node*mob1.tcl
source cbr50*node*.tcl

puts "Start of simulation.."

# Define node initial position in nam
for {set i 0} {$i < $opt(nn)} { incr i } {
# 30 defines the node size for nam
$ns_ initial_node_pos $node_($i) 30
}
```

```
# Telling nodes when the simulation ends
for {set i 0} {$i < $opt(nn) } { incr i } {
   $ns_ at $opt(stop) "$node_($i) reset";
}

# ending nam and the simulation
$ns_ at $opt(stop) "$ns_ nam-end-wireless $opt(stop)"
$ns_ at $opt(stop) "stop"
$ns_ at 800.01 "puts \"end simulation\" ; $ns_ halt"
proc stop {} {
   global ns_ tracefd namtrace
   $ns_ flush-trace
   close $tracefd
   close $namtrace
exec nam dsr50node.nam &
exit 0
}
$ns_ run
```

2. **Penulisan Command Prompt Model Propagasi TwoRay Ground**

```
kevianda@ubuntu: ~/ns-allinone-2.35/ns-
2.35/indep-utils/propagation$ ./threshold –m
TworayGround 300
```

3. **Penulisan Command Prompt Random Mobility Waypoint**

```
kevianda@ubuntu: ~/ns-allinone-2.35/ns-
2.35/indep-utils/cmu-scen-gen/setdest$
./setdest -n 50 -p 0 -M 36 -t 800 -x 1500 -y
300 > dsr50nodemob.tcl
```

4. **Penulisan Command Prompt Lapisan Transport**

```
kevianda@ubuntu:~/ns-allinone-2.35/ns-
2.35/indep-utils/$cmu-scen-gen/ns cbrgen.tcl
-type cbr –nn 50 –seed 10.0 –mc [max.
```

```
connection (2, 4, 6, 8, 10, 12, 14] –rate 1 >
cbr50node.tcl
```

5. Hasil 14 Pasangan Koneksi

```
#
# nodes: 50, max conn: 14, send rate: 1, seed: 10.0
#
#
# 1 connecting to 2 at time 25.561746128630705
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(1) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(2) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 25.561746128630705 "$cbr_(0) start"
#
# 2 connecting to 3 at time 74.10354682901108
#
set udp_(1) [new Agent/UDP]
$ns_ attach-agent $node_(2) $udp_(1)
set null_(1) [new Agent/Null]
$ns_ attach-agent $node_(3) $null_(1)
set cbr_(1) [new Application/Traffic/CBR]
$cbr_(1) set packetSize_ 512
$cbr_(1) set interval_ 1
$cbr_(1) set random_ 1
$cbr_(1) set maxpkts_ 10000
$cbr_(1) attach-agent $udp_(1)
$ns_ connect $udp_(1) $null_(1)
$ns_ at 74.10354682901108 "$cbr_(1) start"
#
# 2 connecting to 4 at time 178.47412263903493
```

```
#
set udp_(2) [new Agent/UDP]
$ns_ attach-agent $node_(2) $udp_(2)
set null_(2) [new Agent/Null]
$ns_ attach-agent $node_(4) $null_(2)
set cbr_(2) [new Application/Traffic/CBR]
$cbr_(2) set packetSize_ 512
$cbr_(2) set interval_ 1
$cbr_(2) set random_ 1
$cbr_(2) set maxpkts_ 10000
$cbr_(2) attach-agent $udp_(2)
$ns_ connect $udp_(2) $null_(2)
$ns_ at 178.47412263903493 "$cbr_(2) start"
#
# 9 connecting to 10 at time 77.022611916541408
#
set udp_(3) [new Agent/UDP]
$ns_ attach-agent $node_(9) $udp_(3)
set null_(3) [new Agent/Null]
$ns_ attach-agent $node_(10) $null_(3)
set cbr_(3) [new Application/Traffic/CBR]
$cbr_(3) set packetSize_ 512
$cbr_(3) set interval_ 1
$cbr_(3) set random_ 1
$cbr_(3) set maxpkts_ 10000
$cbr_(3) attach-agent $udp_(3)
$ns_ connect $udp_(3) $null_(3)
$ns_ at 77.022611916541408 "$cbr_(3) start"
#
# 12 connecting to 13 at time 42.581176470304456
#
set udp_(4) [new Agent/UDP]
$ns_ attach-agent $node_(12) $udp_(4)
set null_(4) [new Agent/Null]
$ns_ attach-agent $node_(13) $null_(4)
set cbr_(4) [new Application/Traffic/CBR]
$cbr_(4) set packetSize_ 512
$cbr_(4) set interval_ 1
$cbr_(4) set random_ 1
```

```
$cbr_(4) set maxpkts_ 10000
$cbr_(4) attach-agent $udp_(4)
$ns_ connect $udp_(4) $null_(4)
$ns_ at 42.581176470304456 "$cbr_(4) start"
#
# 13 connecting to 14 at time 169.44993819550143
#
set udp_(5) [new Agent/UDP]
$ns_ attach-agent $node_(13) $udp_(5)
set null_(5) [new Agent/Null]
$ns_ attach-agent $node_(14) $null_(5)
set cbr_(5) [new Application/Traffic/CBR]
$cbr_(5) set packetSize_ 512
$cbr_(5) set interval_ 1
$cbr_(5) set random_ 1
$cbr_(5) set maxpkts_ 10000
$cbr_(5) attach-agent $udp_(5)
$ns_ connect $udp_(5) $null_(5)
$ns_ at 169.44993819550143 "$cbr_(5) start"
#
# 16 connecting to 17 at time 15.633268074893051
#
set udp_(6) [new Agent/UDP]
$ns_ attach-agent $node_(16) $udp_(6)
set null_(6) [new Agent/Null]
$ns_ attach-agent $node_(17) $null_(6)
set cbr_(6) [new Application/Traffic/CBR]
$cbr_(6) set packetSize_ 512
$cbr_(6) set interval_ 1
$cbr_(6) set random_ 1
$cbr_(6) set maxpkts_ 10000
$cbr_(6) attach-agent $udp_(6)
$ns_ connect $udp_(6) $null_(6)
$ns_ at 15.633268074893051 "$cbr_(6) start"
#
# 18 connecting to 19 at time 166.15229858372001
#
set udp_(7) [new Agent/UDP]
$ns_ attach-agent $node_(18) $udp_(7)
```

```
set null_(7) [new Agent/Null]
$ns_ attach-agent $node_(19) $null_(7)
set cbr_(7) [new Application/Traffic/CBR]
$cbr_(7) set packetSize_ 512
$cbr_(7) set interval_ 1
$cbr_(7) set random_ 1
$cbr_(7) set maxpkts_ 10000
$cbr_(7) attach-agent $udp_(7)
$ns_ connect $udp_(7) $null_(7)
$ns_ at 166.15229858372001 "$cbr_(7) start"
#
# 19 connecting to 20 at time 71.682518502549499
#
set udp_(8) [new Agent/UDP]
$ns_ attach-agent $node_(19) $udp_(8)
set null_(8) [new Agent/Null]
$ns_ attach-agent $node_(20) $null_(8)
set cbr_(8) [new Application/Traffic/CBR]
$cbr_(8) set packetSize_ 512
$cbr_(8) set interval_ 1
$cbr_(8) set random_ 1
$cbr_(8) set maxpkts_ 10000
$cbr_(8) attach-agent $udp_(8)
$ns_ connect $udp_(8) $null_(8)
$ns_ at 71.682518502549499 "$cbr_(8) start"
#
# 22 connecting to 23 at time 139.2243019860351
#
set udp_(9) [new Agent/UDP]
$ns_ attach-agent $node_(22) $udp_(9)
set null_(9) [new Agent/Null]
$ns_ attach-agent $node_(23) $null_(9)
set cbr_(9) [new Application/Traffic/CBR]
$cbr_(9) set packetSize_ 512
$cbr_(9) set interval_ 1
$cbr_(9) set random_ 1
$cbr_(9) set maxpkts_ 10000
$cbr_(9) attach-agent $udp_(9)
$ns_ connect $udp_(9) $null_(9)
```

```
$ns_ at 139.2243019860351 "$cbr_(9) start"
#
# 23 connecting to 24 at time 8.8952016219940049
#
set udp_(10) [new Agent/UDP]
$ns_ attach-agent $node_(23) $udp_(10)
set null_(10) [new Agent/Null]
$ns_ attach-agent $node_(24) $null_(10)
set cbr_(10) [new Application/Traffic/CBR]
$cbr_(10) set packetSize_ 512
$cbr_(10) set interval_ 1
$cbr_(10) set random_ 1
$cbr_(10) set maxpkts_ 10000
$cbr_(10) attach-agent $udp_(10)
$ns_ connect $udp_(10) $null_(10)
$ns_ at 8.8952016219940049 "$cbr_(10) start"
#
# 27 connecting to 28 at time 157.39743432840214
#
set udp_(11) [new Agent/UDP]
$ns_ attach-agent $node_(27) $udp_(11)
set null_(11) [new Agent/Null]
$ns_ attach-agent $node_(28) $null_(11)
set cbr_(11) [new Application/Traffic/CBR]
$cbr_(11) set packetSize_ 512
$cbr_(11) set interval_ 1
$cbr_(11) set random_ 1
$cbr_(11) set maxpkts_ 10000
$cbr_(11) attach-agent $udp_(11)
$ns_ connect $udp_(11) $null_(11)
$ns_ at 157.39743432840214 "$cbr_(11) start"
#
# 27 connecting to 29 at time 83.269324332135412
#
set udp_(12) [new Agent/UDP]
$ns_ attach-agent $node_(27) $udp_(12)
set null_(12) [new Agent/Null]
$ns_ attach-agent $node_(29) $null_(12)
set cbr_(12) [new Application/Traffic/CBR]
```

```
$cbr_(12) set packetSize_ 512
$cbr_(12) set interval_ 1
$cbr_(12) set random_ 1
$cbr_(12) set maxpkts_ 10000
$cbr_(12) attach-agent $udp_(12)
$ns_ connect $udp_(12) $null_(12)
$ns_ at 83.269324332135412 "$cbr_(12) start"
#
# 28 connecting to 29 at time 21.653521415616162
#
set udp_(13) [new Agent/UDP]
$ns_ attach-agent $node_(28) $udp_(13)
set null_(13) [new Agent/Null]
$ns_ attach-agent $node_(29) $null_(13)
set cbr_(13) [new Application/Traffic/CBR]
$cbr_(13) set packetSize_ 512
$cbr_(13) set interval_ 1
$cbr_(13) set random_ 1
$cbr_(13) set maxpkts_ 10000
$cbr_(13) attach-agent $udp_(13)
$ns_ connect $udp_(13) $null_(13)
$ns_ at 21.653521415616162 "$cbr_(13) start"
#
#Total sources/connections: 12/14
#
```

6. File Mac802-11.cc Modifikasi Alonso – Rocha

...

```
// Added by Alonso - Rocha to support received power tracing
#include <vector>  #include <stdlib.h>  ...

void
Mac802_11::RetransmitRTS()
{
...

if(ssrc_ >= macmib_.getShortRetryLimit()) {
```

```
    discard(pktRTS_, DROP_MAC_RETRY_COUNT_EXCEEDED);
pktRTS_ = 0;
hdr_cmn *ch = HDR_CMN(pktTx_);
if (ch->xmit_failure_) {

        struct hdr_mac802_11* dh = HDR_MAC802_11(pktTx_);
        //destinity dh_ra

        uint32_t idNode = ETHER_ADDR(dh->dh_ra);
        double received_Power = pktTx_->txinfo_.RxPr;

        int i = findNode(idNode, nodesPower);
        if (i != -1)
           received_Power =
        nodesPower[i].power[nodesPower[i].pos];
        // designed average mode
        //float av=0; int movAw=0;
        //average(nodesPower[i],av,movAw);

       if (received_Power > RxThreshold )
       {
         ch->size() -= phymib_.getHdrLen11();
         ch->xmit_reason_ = XMIT_REASON_HIGH_POWER;
         ch->xmit_failure_(pktTx_->copy(),
         ch->xmit_failure_data_);

} // Alonso - Rocha ends
else {

ch->size() -= phymib_.getHdrLen11();
ch->xmit_reason_ = XMIT_REASON_RTS;
ch->xmit_failure_(pktTx_->copy(),
ch->xmit_failure_data_);

else {
// Alonso - Rocha begins  // source rf_ta; dest = rf_ra
if (average_selected){
u_int32_t idNode = ETHER_ADDR(rf->rf_ra);
```

```
int pos =findNode (idNode, nodesPower);
if (pos!=-1){
    //tell us if the node is moving away in the last 1, 2 or n
    movements
   float av=0; int movAw=0;
    average(nodesPower[pos],av,movAw);

        //these values need a deep study to improve results
        if ( (movAw >= 18) && (av < RxThreshold) ) {
            //cout <<"LOW POWER (rts) idnode: "<<idNode<<"
        av: "<<av << " movaw: FarFactor:
        "<<av/RxThreshold<<" retryCount: "<<ssrc_<<endl;
        discard(pktRTS_,
        DROP_MAC_RETRY_COUNT_EXCEEDED); pktRTS_ =
        0; hdr_cmn *ch = HDR_CMN(pktTx_);
        if (ch->xmit_failure_) {
             ch->size() -= phymib_.getHdrLen11();
             ch->xmit_reason_ = XMIT_REASON_RTS;
            ch->xmit_failure_(pktTx_->copy(),
            ch->xmit_failure_data_);

        }
        discard(pktTx_,
        DROP_MAC_RETRY_COUNT_EXCEEDED);
        pktTx_ = 0;  s
        src_ = 0;
        rst_cw();

        return;
        }

    }

}

    // Alonso - Rocha ends

inc_cw();
```

```
mhBackoff_.start(cw_, is_idle()); }

}

...
void
Mac802_11::recv(Packet *p, Handler *h)
{

...
if(tx_active_ && hdr->error() == 0) {
hdr->error() = 1;

  }
// Alonso - Rocha
//insert here the packet in our array of nodes and powers
hdr_mac802_11 *mh = HDR_MAC802_11(p);

//dh_ra = dest; dh_ta = source
u_int32_t idNode = ETHER_ADDR(mh->dh_ta);
double power = p->txinfo_.RxPr;
insertNode (idNode,power,nodesPower);
// Alonso – Rocha

...
}
```

7. File Mac802-11.h Modifikasi Alonso – Rocha

```
...

#include "mac-timers.h"
#include "marshall.h"
#include <math.h>

// Added by Alonso - Rocha to support received power tracing
#include <vector>
```

class EventTrace;

...

struct hdr_mac802_11 {

...

};

//Alonso - Rocha: structure for keeping the received power of the *node*s struct time_power {

```
  int pos;
  double power[20];
      u_int32_t id Node;
};
```

#define DSSS_MaxPropagationDelay

class PHY_MIB {
public:
...

Private:

...

// Added by Alonso-Rocha

int find*Node* (uint32_t id*Node*, vect *node*sPower);

void insert*Node* (u_int32_t id*Node*, double power, vect &v);

void initializePowers(time_power &tp);

void previousPos(int &pos);

void nextPos (int &pos);

void average(time_power tp, float& averag, int &movingAway);

// debug procedures

void viewVector (vect v);

void viewArray (time_power tp);

// Alonso - Rocha ends

...

// Added by Alonso - Rocha to support received power storage vect *node*sPower;

// int average_selected; variable that activates average mode

...

8. file packet.h modifikasi Alonso – Rocha

```
struct hdr_cmn {

    ...
    #define XMIT_REASON_RTS 0x01
    #define XMIT_REASON_ACK 0x02
    // ktnahm add for DAMPEN policy #define
    XMIT_REASON_CONFIRM 0x03
    // Alonso - Rocha //cross-layer feature
    #define XMIT_REASON_HIGH_POWER 0x04
    ... };
```

Halaman ini sengaja dikosongkan

BIOGRAFI PENULIS

Kevianda Kamarullah atau biasa dipanggil Kevin lahir di Jakarta, 14 Juni 1995. Mengenyam pendidikan di SD Bakti Mulya 400, SMP Bakti Mulya 400, yang kemudian melanjutkan pendidikan di SMA Al-Izhar Pondok Labu, dan melanjutkan studi S1 di Institut Teknologi Sepuluh Nopember (ITS) Surabaya jurusan Teknik Elektro bidang studi Teknik Telekomunikasi Multimedia. Aktif mengikuti beberapa kepanitiaan seperti Google Campus, Electrical Engineering Event, dan juga menjadi ketua asisten praktikum Dasar Sistem Telekomunikasi untuk Jurusan Teknik Biomedik. Selain itu juga pernah mewakili ITS dalam mengikuti Tech in Asia Conference di Jakarta pada tahun 2015. Cita-cita yang dimilikinya yaitu dapat mendirikan perusahaan teknologi besar yang dapat menjadikan Indonesia menjadi bangsa yang terdepan dalam urusan teknologi informasi dan komunikasi.

Email: kevianda@icloud.com